Ontology guided financial knowledge extraction from semi-structured information sources

by

Eivind Bjoraa

Masters Thesis in
Information and Communication Technology

Agder University College

Grimstad, May 2003

Abstract

Intermedium has an agent searching the Web for financial articles defined by certain criteria, for instance an industrial domain of interest. A portal service for reading and searching these articles, are available for the customers. The sources searched among are secondary sources, like online newspapers. Secondary sources publish information more frequently, and other information than can be found in annual reports etc, like predictions. Finding and comparing financial figures in the articles are often time consuming and hard to compare with each other. Having the financial figures, and what these applies for, presented in an application where information could be easy reviewed and compared, would apply valuable information for decision makers in bigger companies.

Web documents are usually semi-structured, and therefore almost impossible to query for information. Only keyword searches are supported by the computers because of the lack of understanding. Advanced extraction processes of the information needs to be performed. This thesis evaluates an ontology guided approach for extracting financial information from semi-structured information sources.

A financial ontology has been constructed based on an investigation of 50 articles gathered from Intermedium's agent. Instances with synonyms, the words to extract from the text, and relations between the instances have been defined. The ontology language RDF has been chosen and used as ontology language through the entire thesis.

A prototype application has been developed to perform the extraction process. Articles are loaded from XML files; words to extract from the text are found by query the ontology using the query language RDQL; NLP and NLTK are used to do the extraction based on the words found in the ontology; Velocity template is used to get the proper structure in the output files RDF and XBRL instance document. The ontology is providing the application with knowledge in the extraction process. When a synonym is found in one instance, a query for reference to other instances is performed, and synonyms of these instances are searched for in the text. If a text does not contain any interesting information, the application does not waste time with trying to match all words in the ontology with the ones in the text.

The result is presented with semantic tagging in RDF syntax. A part of the information extracted is also shown as an example of how the financial standard XBRL can be given. The advantage of XBRL is that it can be used directly by supporting tools; RDF has to be processed by a more intelligent application. Financial information has in both these formats been added knowledge with computer processable semantic tagging.

Preface

This thesis is written for Intermedium, Grimstad, and is part of the "Master in Technology" degree at Agder University College. The work has been carried out in the period between December 2002 and May 2003.

I would like to thank my supervisors, Vladimir Oleshchuk at Agder University College and Asle Pedersen at Intermedium, for valuable help and inspiration during the entire process of this thesis.

Grimstad, May 2003

Eivind Bjoraa

Contents

1 Introduction

1.1 Background

Users of the Word Wide Web have the last decade seen a dramatically increase in electronically available information. Looking at the Internet as a massive information repository, a major part of this information lies available from every online computer just some keywords away from our computers.

However, finding the right information at the right time is not always a trivial job. Web agents of several kinds have been developed to help us solve different issues, including search engines. Google and Fast are two agents for finding information on the Web. Although many of these available agents are doing a good job, they do not speak the same language as humans. They are restricted to keyword based techniques [Fen02], which reduces their efficiency. We can therefore not ask Google to find information about all cars made by BMW; rather we need to know what to search for and specify this in our search statement, like for instance "BMW 320, 325, 520". This search may also find hits we were not interested in, and thereby making it necessary to make an additional human selection.

Things are however about to change in the area of search engines. Google have recently, at the end of April, acquired "Applied Semantics". This company produces software applications for semantic text processing, and online advertising to understand, organize and extract knowledge from websites. Finding information without any human browsing needed, where intelligent searches provide answers based on search questions [Fen02] like the BMW example above, are more focused. This is also what Google tries to achieve by acquiring Applied Semantics.

The problem of doing intelligent searches up to now has basically been a lack of semantic markup of information on the Web. Much of the information is provided only for human consumption in form of natural language [Ala03], which is not understandable for computers. Adding semantics to Web pages are however seen as a tremendous amount of work. Humans can not reach over it and automation may not always provide the right semantic tagging because of lack of understanding of the information given [Gan02].

Ontology has been adopted into computer science from the area of philosophers', where it is used to describe words as they exist and the relations between them [Gru93]. Ontologies can provide a way around the lack of semantic tagging of information. Some major projects have tried this approach ([Ala03] and [Fen02]) for gathering and managing of information from Web or given documents. These are described more in Section 4.2.

These projects gap over many areas and functions. This thesis will look into financial information which can support decision makers at the administration level of companies or in some financial domain. Leaders need to have the right and all wanted information easy accessible to support them to make the best decisions. Relevant information may for instance be financial reports, predictions and statements publicly available on different company or news web sites. This information is available for everyone as long as you know where to find it. However, these press releases and reports are often large text files which are time consuming both to read and to find useful information in. Like in the BMW example above, you need to make a human selection of information. A preferable solution would be to have a web agent finding the information you want, specified by certain criteria's, independent of the expression used to provide this information. At the end, the information is represented in a structured way which is understandable and processable for computers.

1.2 Thesis environment

Intermedium has, as a service provider of competitive intelligence, developed different agents. One of these is a Web search engine searching news and press releases based on given criteria, like domain and companies. This information is gathered from several different Web sites and displayed at a single page individual for each company. Lists of news are displayed with information about the article divided into date, subject with a short heading of what the article is about, criteria and link to source.

I have been given access to the sites for two of the companies using this service. This enables me with real world data to base my practical work upon.

1.3 Thesis definition

Thesis subject definition

"There is available semi-structured data stored in a database as plain texts or html pages. The purpose of this project is to design new methods for extraction of data and, based on this, develop a prototype for extracting financial information from the semi-structured text. For example, in the financial world numbers are often one main target, but they are meaningless without any semantic meta-data describing what kind of information they represent. It is therefore necessary to extract semantic information about the numbers, like currency and financial description, from the text in which they occur.

The main challenges in this project are trying to design a financial ontology and use this in the prototype to assist the extraction and to provide better knowledge. We plan to use an ontology description language, like OWL or Topic Maps, to explain the extracted entities. The extracted knowledge should be captured in such a way that it can be exported into financial standards like XBRL."

Goals and objectives

- *Make a survey of ontology based techniques and methods*
- *Design a smaller, or find a suited, ontology for the financial domain.*
- *Develop a prototype for extraction of financial information.*
- *Use the ontology in the prototype to provide better knowledge in your prototype.*
- *Capture the financial knowledge in a standard financial language like XBRL.*

1.4 Thesis work

Currently, the two major projects, Artequakt [Ala03] and On-to-Knowledge [Fen02] have reached further within the area of ontology guided information extraction than other projects. The Artequakt project has implemented an online Web site gathering biography of artists. It automatically extracts information about the artists from Web pages and populates a knowledge base.

The On-to-Knowledge project has during the process developed many professional tools, purchasable for a fee, used in their projects. They also have big customers, like Swiss Life and BT, where fast and reliable access to long documents, and more efficiently dissemination of customer-handling rules are provided. If the ontology editor and other tools had been freely available, these would have probably been used in this project, claimed by Ontoprise[1] to be the most complete modeling tools.
Ontology is a quite new method of thinking in computer science, and has gained a lot of interest in recent years. This thesis will; through constructing a financial ontology and developing a prototype application; find out whether an ontology guided approach for extracting financial information from semi-structured sources can capture financial information in a semantically tagged way. The result will be financial knowledge expressed in an ontology language and the financial standard XBRL.

A conceptual solution will be presented, proposing a way to solve this issue, exemplified by giving an optimal result in RDF and XBRL.

An ontology, defined by [Ber01], has been constructed for the financial domain. A prototype application have also been developed using the specified words and relations in the ontology to extract financial information from semi-structured sources. Natural language processing and the NLTK toolkit have been used in the extraction part.

[1] Ontoprise – http://www.ontoprise.de/products

1.5 Report outline

In the following chapters, a closer look at the techniques mentioned above will be presented. In chapter 2, a brief description of why the need of information capture is of importance, and background of important information source and why these are so, will be given. Chapter 3 introduces techniques and methods used to enable ontology guided financial extraction from semi-structured information sources. This chapter is divided into four main parts; Semantic Web as the background for the next Section; Ontology; Natural language and processing of this; and at the end an introduction to the financial format standard XBRL. A brief survey of related projects in the area of ontology guided information extraction is presented in chapter 4, followed by a conceptual solution exemplified by an optimal scenario in chapter 5. In chapter 6, the process of constructing the financial ontology, the application prototype for extracting information, including an algorithm for how this process is carried out, is presented. The output in RDF and XBRL are also presented in this chapter. In chapter 7, the results are discussed, before a conclusion is given in chapter 8.

2 Web agent and information capture

2.1 Introduction

Intermedium is providing an automated competitive intelligence service for different companies. A separate Web agent searches the Web for online news articles and press releases for specified domains for the customer companies. Although companies operate in different domains, there are many similarities about their informational needs. The two companies studied, have both a desire and need to keep track on trends and news concerning their business domain. This includes new products developed by competitors and financial news, like stock trends and annual reports, about other firms in their domain.

This agent finds many news articles of information for these companies and their domains. As an user of this agent you have the possibility of listing all news hits, or limit hit results by selecting profile; domain dependent criteria's; sources where the news are selected from; and the time factor from which week the news where published. All these properties make it easier for the user to limit and find wanted article.

The list of articles found are presented with information about every of the filtering option I stated above. These including date published; subject, which are the ingress of the article; which criteria it matches; and from which source it is taken from. The header in the subject field is deployed as a link to the source where you find the article and the source provide a link to the main page of the news provider of current article.

2.2 Weaknesses with current information presentation

Financial information, like annual reports, is regularly publicly released. For instance are annual reports only released once a year and does describe real figures like actual results, for the company this period. Retrieving information more than once a year, or a limited number of times, makes it hard to stay up to date of competing companies' status and its financial movements. Secondary sources, like online newspapers and magazines, are publishing articles which occur more frequently than the reports the companies present themselves. A chance to be more continuously updated in the domain is possible through these sources. Despite to the reports, like annual reports, which provide all financial figures for the current year, secondary sources provide résumé's or information that never or rarely is found in reports. Examples of this is refined data; predictions and forecasts of e.g. financial results due to income or loss; financial analyses of results; forecasts etc.

Intermedium's agent supports the user by finding interesting articles defined by filters. One of the restrictions is not that they should contain financial information. But after browsing through an amount of articles, it is discovered that many of them do contain useful information. This was especially true around December and January, where lots of reports and results were presented. In April and May fewer articles contain financial information.

Consuming this information is, however, not so easy. When you open a link from the default site, a new browser window with the selected page appears. Several Internet sites today get their main income from advertising, making Web pages information intensive. The window is filled with links, banners and pictures, which sometimes are moving. All these advertisement factors strive to get your attention while you focus on getting some information from the article.

Ignoring advertisement, getting the core message of an article can still require an effort. Information, like annual financial result, loss predictions and share changes, are often interlaced with textural expressions forcing the user to read through the entire or major parts of the article to obtaining the desired information. This is time consuming and makes it hard to utilize all opportunities of the information found. This often leads to the fact that it is hard to find the right information when you really need it.

Having the optimal agent finding all the information you want, a tool could have presented this in a way easier to conceive than long articles where finance data needs to be searched for. Different queries and ways of presenting the result could have been available. In addition could integration of different information concerning a company be presented together, or same information compared for validation.

Solving the optimal solution starts out by locating data given in articles. When doing this a problem occurs. Web pages in general contain very little semantic meaning of the text, also referred to as semi-structured text. Some structure is defined and allowing simple queries for retrieving information. Unstructured Web pages has no standard organization and no precise relations among the data given are made. Many of the articles provided by the agent can be placed in between semi- and unstructured. The term weakly structured, used by [Fen02], can give a good description of the article retrieved by the agent.

Another major problem is that, as we operate with weakly structured information, is that there exists no standard way of saying, or writing, the same information. For instance can "estimated income next year to be 5 million dollar" and "current expectations for next years revenue are $5.000.000" refer so to speak to the exact same information. Words have synonyms used, while numbers and currencies can be written in different compounds of numbers, words, abbreviations and signs. The opposite problem occurs when different information is written equally, called ambiguous words, like the word golf

used both for the car and the sport. In finance a share can denote both the verb for sharing and a share at the stock exchange. As long as the information is weakly structured, no metadata are added to the text telling us that golf is a car or the sport. To find out what the word refer to, you need to analyze the words context.

2.3 Enhancing Intermedium's agent

Extracting information from weakly structured sources can not be done by simple queries, like querying databases or XML files. To minimize information to concentrated knowledge given to the user, more advanced techniques have to be applied. Extraction can be done by natural language processing, but a definition of what to extract has to be made.

For further development of Intermedium's agent, I will try to solve some of the problems noted in Section 2.2. I will develop methods to extract financial information from the articles found by the agent. Guided by ontology in the extracting process, groups of synonyms will be defined and given the same meaning by assigning information to a standard tag language and thus providing the information a structure. This will enable the user to concentrate on the financial measures, enabled for querying, more suitable representation and ability easier comparing information from different articles.

In the next chapter, background theories used for developing an agent extracting financial information are presented.

3 Semantic Web and Ontology

3.1 Background

Already at the 1970's the need of a standardized way of encoding knowledge was recognized because of the lack of merging and sharing information between projects. A proposal made by the American National Standard Institute (ANSI) stated that an application domain should be collected in a single conceptual schema [Klu]. Figure 1 illustrates an integrated system where each circle has its own specialization, but they have all a common application knowledge represented in a shared conceptual schema. "*The user interface calls the database for query and editing facilities, and it calls the application programs to perform actions and provide services. Then the database supports the application programs with facilities for data sharing and persistent storage. The conceptual schema binds all three circles together by providing the common definitions of the application entities and the relationships between them.*" [Sow01]

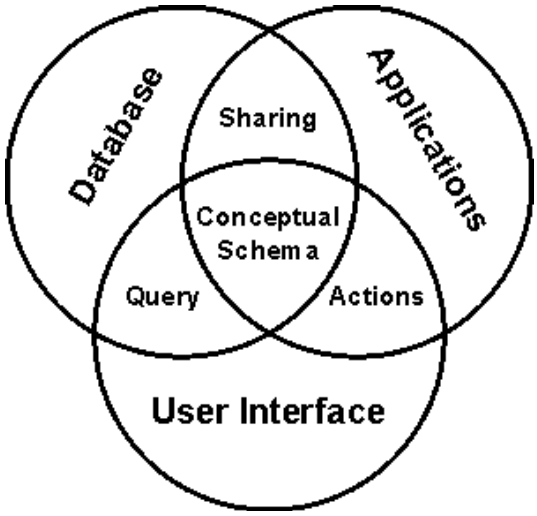

Figure 1: Conceptual schema as the heart of an integrated system [Sow01].

Since the 70's has a shared conceptual schema played an important role in integrated application design, development and use. Several developments have been made based upon this schema. Among these are the fourth generation languages (4GLs); the object-oriented programming systems (OOPS); and tools for computer-aided software engineering (CASE). They have all solved some problems, but integration around a unified schema has not yet succeeded. One of the latest attempts is to integrate knowledge on Web into the Semantic Web, also referred to as the second generation of the World Wide Web. Semantic Web is presented in the next Section.

3.2 Semantic Web

3.2.1 Introduction

Before describing the Semantic Web, a problem scenario is presented describing the problem of current Web and what the Semantic Web strives to solve.

The Web has expanded to a huge repository of information, with an innumerable amount of sources and links between them. One problem is that the information has mainly been published for human consumption, which results in that although the information is out there; there is not always an easy way of finding what you are looking for. Several Web search agents, like Google[2], provide a quite good job searching and identifying potentially good candidates from billions of Web pages[3]. However, as the engines still are keyword based and have no machine understanding, is the query result restricted significantly. Take example where you are looking for a book or paper written by Mr. Jones. The search engines tend to discard different words you may have written in your statement, like *about* or *by*. The result of the often returns loads of links and references referring to books or papers of Mr. Jones, and not only the actual article or book searched for.

What if different information was related? Like a book or paper written by a person object, Mr. Jones, with a relation between them named *author* or *written by.* Then a search could be narrowed the result to actually books and papers written by Mr. Jones. Another example is a search for *palm*. Current search engines do not know if this term refers to a company, the operation system or the PDA named palm. The result of this search will display hits of all these occurrences, and make it necessary for human browsing of the output to find the right result.

A statement by Gandon points to the problem of current Web:
"*The World Wide Web was originally built for human consumption, and although everything on it is machine-readable, this data is not "machine-understandable". Because of that, it is hard to automate things on the Web, and because of the volume of information it is not possible to manage it manually*" [Gan02].

[2] Google – http://www.google.com
[3] May 1st - 3.083.324.652 searched pages by Google.com

3.2.2 The Semantic Web idea

The problem of the huge amount of information mainly designed for human consumption can be solved by associating meaning with content, enabling computers to understand and process information. This idea, named the Semantic Web, was first introduced in 1996 by Tim Berners-Lee [Ber96], inventor of the World Wide Web. He describes it as follows:

> "*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*" [Ber01].

From a high perspective, we can say that the current Web is a provider of pages containing information and links between them designed for human consumption. The Semantic Web aims to augment the existing human-readable Web by adding machine-readable descriptions to Web pages and other Web resources [Hen03]. This is accomplished by giving the information an explicit well-defined meaning, also referred to as metadata, or data about data. Information is considered as a resource [Ste00], where each resource can be linked to any other resource uniquely identified by its URI[4], as specified in [Ber98]. URL[5], a Web link, is the most commonly used URI.

The Semantic Web's ability to reference and identify resources is one of its foundations and makes it look like a great global mesh, or a big global database. Berners-Lee describes it as "Weaving the Web" in [Ber99]. Utilizing this global mesh, software agents will be able to roam from page to page readily carrying out sophisticated tasks for the users [Ber01]. The vision is to be able to make a query where an agent gather information from the Web and provides the right and full information, even though it is presented at different pages or sources.

3.2.3 RDF: Enabling the Semantic Web

Through the Semantic Web idea, standardizations by W3C have been made to facilitate semantic interoperability, enabling exposing and processing of metadata. This standard is a W3C recommendation called RDF, Resource Description Languages [W3C99rdf], based on the syntax of XML[6]. An important reason why this format is chosen is that XML lets you create your own tags. Only following the few rules for tagging, the structure can be built quite freely. XML also provide the important facility of namespaces which precisely associate each property with a schema defining its property. This is in fact the meta-data associated to information.

[4] URI - Unified Resource Identifier [w3Adr]
[5] URL - Unified Resource Locator
[6] XML - eXensible Markup Language, see more at [W3Cxml]

The structure in XML itself does, however, not specify any relations between the tags. RDF is an extension of XML, which encodes the meaning in sets of triples, where each triple is a resource, a property and a value. Almost like subject, predicate and object of an elementary sentence in the human language. These three words are also often used to describe the RDF triple. Inserting values in this triple can be done like this; a resource can for instance be the book *Hamlet*; the resource has a property (or predicate) like "*author*" with *William Shakespeare* assigned as value (object or literal).

In RDF all the triples are identified by an URI each, introduced in Section 3.2.2. This enables anyone to define a new resource, predicate or object, simply by assigning an URI for it. Encoding information with URI ensures a uniquely defined meaning where resources are not only words, but tied to a unique definition [Ber01]. There are however no assumptions that the URI refers to any useful Web page or even refers at all [Mae02].

The resource to be described in this example is the book *Hamlet*. The property (or predicate) is a slot defining relationships between a resource and an object. In this example the book *Hamlet* has a property *author* referring to *William Shakespeare* given by the object. Put together the RDF triple and get a statement as follows.

http://www.books.com/Hamlet HAS author William Shakespeare

Expressed in RDF syntax this statement will be readable and understandable to machines, and look something like this:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.books.com/Hamlet">
    <s:author>William_Shakespeare</s:author>
  </rdf:Description>
</rdf:RDF>
```

Table 1- RDF/XML Syntax

To RDF files, there are always a belonging RDF Schema, RDFS. In table 1 is meta-data given through the two namespaces declared as "rdf:" and "s:". The namespace "s:" is a reference to this file's RDF Schema. RDFS can be seen as a dictionary uniquely defining and restricting terms used in RDF statements. Rules for using RDF properties are described. The namespace "rdf:" points to the W3C standard used in this RDF statement.

Giving information meta-data provides machines the ability to read and understand it. A Web search for books Shakespeare is author of, will provide us with the information Hamlet and where to find it with URL.

Reading large files of RDF in this format provide no easy way to get an understanding or a quick overview of RDF expressions for humans. Therefore the RDF standard [W3C99rdf] provides a standard of drawing diagrams and figures to visualize the expressions. Ovals are representing resources, while arcs represent named properties and rectangles string literals. Resources and literals are also seen as nodes. This way of visualization the statement, also called graph, provides a good way of understanding relations defined in RDF, it is referred to as a graph. The graph for the statement above is given in figure 2.

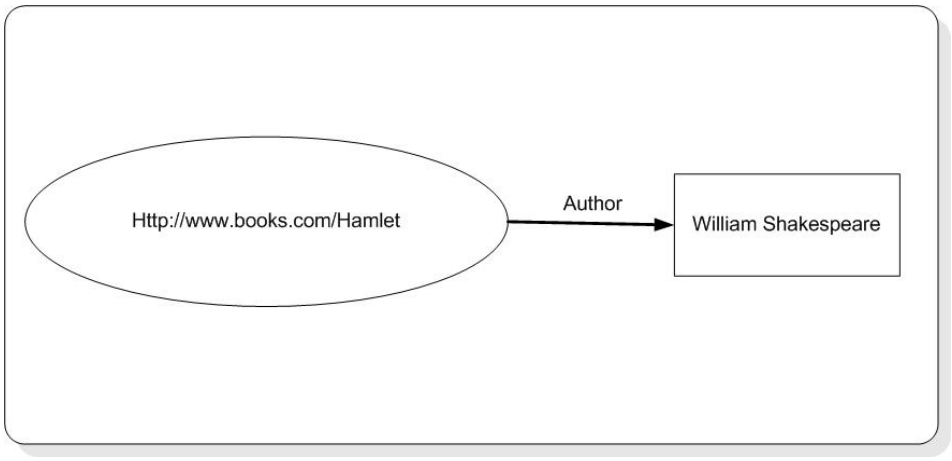

Figure 2 – RDF statement graph

This example can easily be extended by for example adding more information about Shakespeare, like year of birth and death, and other books he has written. Resources can, as described in Section 3.2.2, have relations to other resources connected by a property. For more information, the recommendation by W3C at [W3C99rdf] is referred.

There are in fact three kinds of nodes in RDF; literals which are a string, URI nodes and blank nodes. As URI is an unique identifier, RDF thereby assumes that two nodes with the same URI are the same resource. Blank nodes have no URI label; to discover which resource they represent, the statements assigned has to be looked at.

RDF is, as mentioned above in this Section, an extension to XML, where only hierarchical nesting is providing some sort of relations between words. Since RDF supports unique definitions of words, and relations between them, can this be used in projects concerning information objects and relations between them, almost like a kind of vocabularies. Example of projects are the vCard project [W3Cvcard], electronic business card profile, and FOAF (friend for a friend) for managing online communities [foaf03].

Some tools for adding Semantic information on Web are available. One example of this is Simple HTML Ontology Extensions[7], SHOE. SHOE provides Web page authors the ability to annotate Web documents with machine-readable knowledge. Another project, Dublin Core[8], is developing standards for adding more descriptive records, meta-data tags, to information resources. These information resources range from online pictures to Web pages and online documents [Hil01].

The problem by adding meta-data to Web pages is that it requires some extra effort compared to just publish information the "old way" and it is therefore still lack of use of semantic representation of information on Web. Manually updating existing Web pages will also be a too tremendous job for humans [Gan02], so an automatization may be the only solution. Precision of the result provided can be a problem with this approach. Constructing general agents doing this task may produce errors or not see or understand all details given by all the different pages. Incomplete results may be produced.

Still the Web is not semantic; this approach can therefore not be used without any solutions providing a definition of words. Ontologies are the link between current Web and the Semantic Web by defining and specifying meaning and relations between terms.

3.3 Ontology introduction

3.3.1 Definitions and basics

The word *ontology* comes from two Greek words, *onto* for being and *logos* for word. Originally the term was used by philosophers to deal with nature and the organization of reality [Gua98], or to model things as they exist in the world.

In the early 90's the term ontology was adopted for use concerning information management and in context of the Semantic Web. Several definitions of the term ontology have since then been made and used within this area. According to Gruber's definition of ontology, we say that;

"*An ontology is an explicit specification of a conceptualization*" [Gru93].

Talking about conceptualization in this context refers to ontology as an abstract model of a particular domain of knowledge. With explicit specification Gruber means that the concepts, their attributes and the relationship between the concepts are given an explicit definition in the ontology, as shown in figure 3.

[7] SHOE – http://www.cs.umd.edu/projects/plus/SHOE
[8] Dublin Core – http://dublincore.org

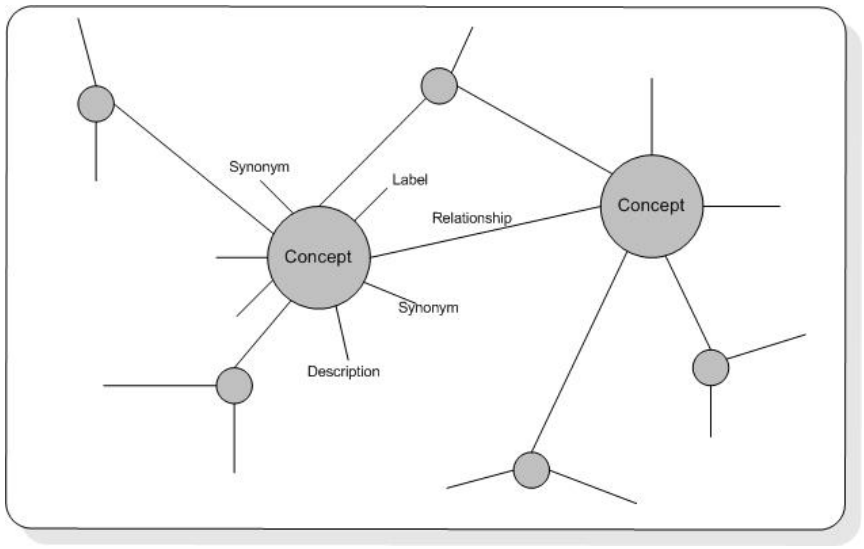

Figure 3 - Conceptual model [Lau02]

By constructing and using ontology, you have made an agreement for use of a known and shared vocabulary of a particular domain of interest for humans, databases and applications. A foundation for communication between humans and machine agents has been established.

The relations between the concepts can be like "subclass-of", "instance-of" or "refers-to" together with functions that give certain information about a concept. An example of this is given in the following function. This says that if we an instance is both a person and a mother; then we know she is a woman.

$$mother\text{-}of \ \& \ person \rightarrow Woman$$

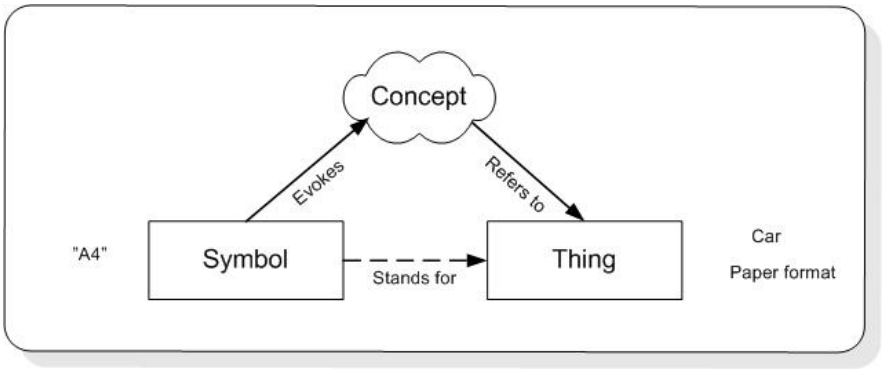

Figure 4 – Meaning triangle [Ogd23]

Concepts are usually grouped into a hierarchy of classes, where each class, and sub-class, can contain sub-classes. Relationships can consequently be established either between different concepts or classes of concepts. Instances in ontology represent a concrete instantiation of a particular defined class.

3.3.2 Ontology example

Consider an ontology for vehicles, containing boats, motorcycles and cars. There exists many manufacturers of cars (e.g. Volvo), and they make different types of cars (S40, S60 and V70). Volvo, BMW and Audi represent different sub-classes of cars in the ontology of vehicle, while V70, 320 and A4 are specific instances of cars made by different manufacturers.

Giving this ontology more information and instances, the ontology will provide a standard, unambiguous representation of the domain of vehicles. When "the symbol" A4 occurs, all users of this ontology will automatically know that this is an instance of the car Audi, because of the instantiation of the concept in the ontology [Das02] [W3Cowl], and not the paper format, see figure 4. If two manufacturers get together to produce an instance car, it would be represented as a relationship between the manufactures. Part of the example is presented in figure 5, where it can be seen that S60, S60 and V70 are the only instances of cars Volvo make.

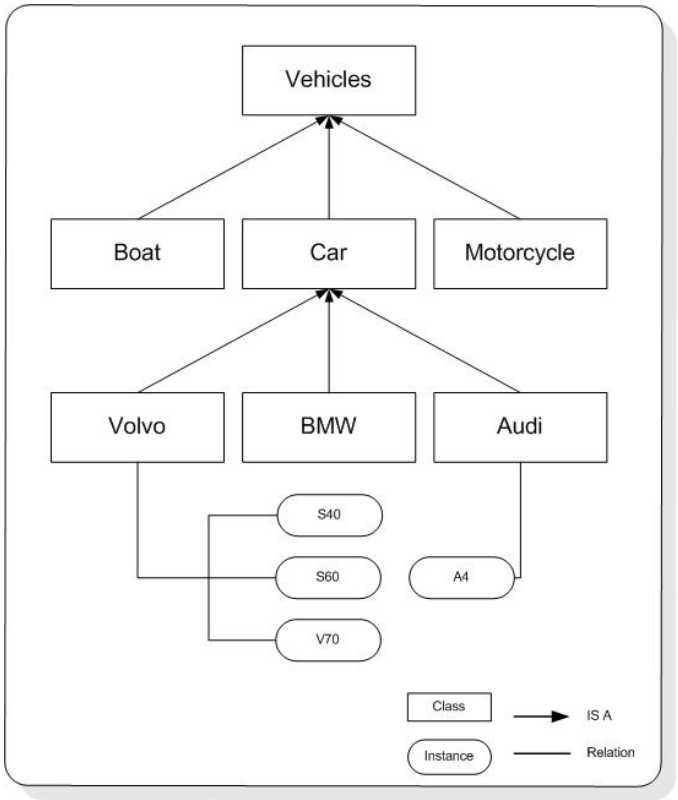

Figure 5 –Ontology example

An important point, also partly reflected in the given case, is that there is m to n relationship between words (symbol) and concepts. Different words may refer to the same concept, or a word may refer to several concepts, like A4 [Mae02]. Ontologies are specified to be a conceptualization of a specific domain, which reduce the amount of these examples to a minimum or to exceptions which can be dealt with separately.

Using an unified ontology as an interface to extract information from heterogeneous distributed data sources, the time-consuming process of learning the individual data sources can be by-passed [Das02]. Ontologies also make the application more flexible and intelligent, in the sense that they can more accurate work at the human conceptual level not depending on any other structures and standards. In the next Section I will look into the some issues concerning construction of ontologies.

3.4 Ontology construction

3.4.1 Introduction

Constructing an ontology can be done in three different ways. Manually construction makes use of one of the many supported tools which in the past few years have occurred[9]. This way requires a great amount of work and getting a proper ontology also often require a domain expert, especially considering that the better an ontology is constructed, the better will it work. The other is a semi-automatic way where a combination of manually construction and additional ontology learning. The third method is automatic creation of ontologies. The advantage is that these to do not require so much effort, and relearning is relatively easy. They are however requiring much more background knowledge and are far more complex to construct.

3.4.2 Top level ontologies

When starting to construct ontologies, you have to decide which level your ontology is aimed for. Guarino states four different types of ontologies; Top-level which deals with very general concepts like time, space and event designed for large communities of users; domain ontology for a certain domain specified in the top-level ontology, like vehicles; task ontology describes vocabulary related to a generic task or activity, like selling, also defined in the top-level; and application ontology as the most specific ontology depending on both a particular domain and task [Gua98], see figure 6.

[9] List of ontology editors - http://www.daml.org/tools

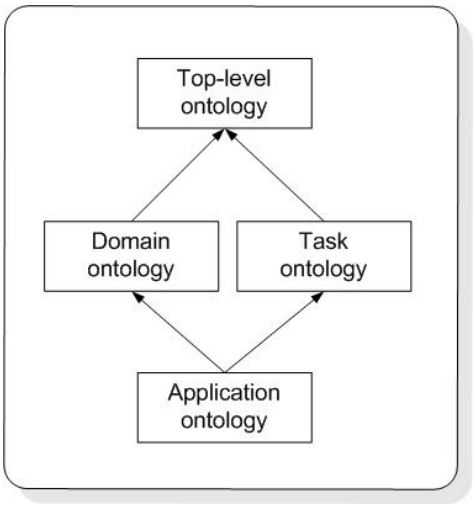

Figure 6 – Types of ontologies [Gua98]

The most important is, however, that the ontology constructed is following the specified needs. If an ontology is made to large, containing a lot of classes and relationships, this may reduce the effort of the ontology. Having to many concepts may result in problems when equal terms appear. Like A4, if both Audi A4 and the paper size A4 are in the ontology, which meaning is the right one for this instance? Making the ontology to narrow, useful information may not be conceived.

It is however not always necessary to start making the ontology from scratch every time. Existing ontologies can be extended by linking or mapping (merging) them together into one. Both methods require some equalities providing right linking or mapping to be performed. I will not go deeper into these topics; instead refer to other resources, like [Doa02], for more information.

3.4.3 Ontology languages

Ontologies do, like Semantic Web, have languages for representation. In fact is RDF/RDFS one of these ontology languages. Ontology languages have, like many other programming standards and languages, evolved through the recent years. Among the early languages are Ontolingua and LOOM, to later ones like RDF/RDFS. The most used at this time are probably DAML+OIL followed by RDF/RDFS. DAML+OIL is a joined force of DAML, an RDF Schema based language, and OIL for better performance [W3Cd+o]. OWL is a language derived from DAML+OIL, and is currently a working draft at W3C [W3Cowl]. All these are based on the previous ones and extend their properties. Key features of OWL, compared to the previous languages, are that it adds more vocabulary for describing properties and classes. Examples of this are new relations between classes, like disjointness, cardinality, equality and richer typing properties. OWL is only supported by a very few number of the currently available ontology editors, because of still being a working draft. There are converters free online transforming your

ontology into OWL. In addition to these languages mentioned, there are several others including Topic Maps, which is briefly looked into, as a comparison to RDF, in the next Section.

3.4.4 Topic Maps

Semantic Web and ontology is driven by W3C, while Topic Map is an ISO standard. Topic Map is also a kind of ontology, like thesaurus, taxonomies and semantic nets also are. Topic Map describes knowledge structures, electronic indices and classification schema to manage the information glut and structuring of unstructured information.

Garshol has made some studies about Topic Map and different ontology languages, like RDF, DAML, OIL and OWL [Gar03].

Topic Map is, as RDF, using XML for structuring information with meta-data in hierarchical manner. They both represent pretty much the same, but uses different names about it. RDF is built upon the triple of resource, property and object. Topic Map has something very equal to this; names, occurrences and associations.

Names, associations and occurrences are important in Topic Map, where RDF is limited to only defining assertions (statement) for things, corresponding to names in Topic Map. Topic Map has different kinds of relationships and associations are, completely different to RDF statements, assigned roles.

Where RDF only allows blank nodes, Topic Map also allows blank URI nodes. This is because URI in Topic Map can be considered to be a subject address (subject identified is the resource) or a subject identifier (subject is whatever is described by the resource). These differences make it hard to achieve interoperability between Topic Map and RDF.

There are also some other differences, about reifications and qualifications. I will not go deeper into this, and refer to more differences and complete comparison at [Gar03].

3.4.5 Ontology editors

Several of the ontology editors available today, especially among the freeware, are developed by an university. Examples of these are Protégé by Stanford University, OilEdit by University of Manchester and Kaon by the University of Karlsruhe. These universities are among the groups who are very active in this field. This is reflected in the papers and project performed, and in updating rate of the editors, functionality and different plugins they support. I have joined a mailing list for Protégé, which is very active and updates are frequently.

Protégé standard download supports construction of ontologies in the following three formats; standard text file; JDBC database; and RDF Schema. The text file format represents one file containing a textual ontology without any URI and a file for instances. The ontology is built hierarchical with classes, subclasses with its properties, constraints and relations described. The instance file is similar to the N3 notation format used in RDF, but with a Protégé syntax variant. The JDBC format of Protégé lets you make your ontology as a knowledge database. Instance file and JDBC part of Protégé have not been further investigated.

One drawback of Protégé is that it does not support OWL, the new ontology language. There are very few who does, since this is a new language not completely finished, but OilEd is however one of them where who support OWL. In OilEd you can export the ontology to a list of different languages. This tool was not chosen because of poor user interface and documentation.

There are many plugins available for Protégé. One of these is support for the ontology language DAML+OIL provided as a third party component from SRI[10]. This plugin was unfortunately not found before it was too late. An ontology in RDF was already constructed, different tools were found and adjusted. I should later be aware after testing, mailing to SRI and response on Protégé's mailing list, that this plug was not working properly and improvement were worked on to next release.

3.4.6 Parsing and querying RDF

Several tools and APIs support querying of ontologies in different formats. Hewlett Packard (HP) has developed a Java API for RDF called Jena. Subsystems of Jena include; ARP Parser; RDQL, RDF data query language; DAML API; and two ways of storing Jena models in relational databases. In addition to querying, Jena can also be used to add triples to a model. Protégé has also a support for querying ontologies.

RDQL defines a query language, which looks like SQL statements used for database queries, for retrieving sets of variables. RDF is treated as data and a query of the triple pattern are provided.

3.4.7 Ontology learning and pruning

Developing ontologies is a time-consuming and error-prone task. Constructing ontologies in a semi-automatic way may reduce the time, and hence some of the money required, especially for maintaining and updating it. Construction of ontology semi- or automatically, together with ontology learning and pruning, are relatively new areas of research. Only the last two, three years projects around this has been made. The

[10] SRI – http://www.ai.sri.com/daml/DAML+OIL-plugin

Artequakt [Ala03], presented in Section 4.2.1, also points to this as an area of further work providing difficulties due to decrease of precision and recall of the ontology.

Maedche and Staab have done research projects in ontology learning, a semi-automatic process supporting ontology engineering and management. Four different phases have been given in [Mae02]; ontology extraction, reuse, evolution and interoperability.

Ontology extraction from scratch is based on the input data to the ontology, and engineers' gets proposals from for ontology modeling tool. Learning from text has for several years been done by Natural Language Processing, see Section 3.4. This has for instance been done by matching patterns. Concept learning is another approach where a given taxonomy is incrementally updated as new concepts from real-world texts are acquired.

Reuse of whole or part of existing domain ontology for a specific application, see figure 6, is known as ontology pruning. Ontology pruning is data-driven and uses an approach that is based on word or concept frequencies. The occurrence frequency are analyzed, and words that refers to concepts in the ontology but do not occur in the domain-specific corpus are eliminated from the learned ontology. This method may also be used for evolving ontologies where words or concepts become unused.

Learning from schema can also provide valuable input for developing an ontology. Many different types of schemas can be used in modeling a system, but almost every system have at one of the following or another schema assigned; UML, XML-schema, ER-models or Document Type Definition. Reverse engineering can then be used as a part of the development process.

Ontology evolution is done by refinements or changing discovery e.g. concept drifts. Ontology interoperability is done be merging or mapping ontologies. Ontology learning from interoperability is accomplished by finding semantic mappings between similar elements from two ontologies and then using both ontologies as one.

Having an ontology provides us with a controlled vocabulary, or shared understanding, describing entities or concepts and how they are related for a certain domain of interest. However, this is not enough, we also need to read through text files and find out what information they give. To achieve this, Natural Language Processing (NLP) is commonly used. NLP is introduced in the next Section.

3.5 Natural language

3.5.1 Introduction

Information is often given in natural language written for human consumption in the way we speak, and not for computer understanding. This is the same problem mentioned in the introduction, in Section 1.1, 2.3 and 3.3.4. Natural language is recognized by lack of tags assigned giving the text any semantic meaning. For computers to understand the natural language, it has to be processed. Natural Language Processing (NLP) is one computational technology where computers process written, and also spoken, language to locate and extract information. NLP attempts to reproduce the human interpretation of the information in a way that computers understand. Patterns in grammar and conceptual relationships between words are key issues in NLP. The goal is to express or match information given by symbols, relations, and conceptual information so that computers can use it to implement an artificial language interpretation [ITw00].

3.5.2 Natural Language Toolkit

The Natural Language Toolkit (NLTK) is actual a toolkit developed for a student course at the University of Pennsylvania [11] providing a basic infrastructure for building NLP programs. It is basically a set of open source modules written in and for the Python programming language. The infrastructure contains a collection of basic classes for representing data relevant to processing of natural language and interfaces for encapsulating the resources and methods needed for performing specific tasks. NLTK includes, among others, interfaces for tokenizing into smaller text units and tagging text as modules. These two will be the most important ones in the extraction process taken place in the *agent* described in chapter 5.

The simplest way of representing a text is as single strings, which are dealt with in the context of semi-structured or weakly structured information sources. Processing text in this format is a very difficult task. By splitting a text into a list of tokens, you will often get a more convenient way of working with the text. This task, converting the string into a list of tokens, is known as tokenizing. The token module processes a text into individual elements of text like words, lines of text or sentences from a text file. Using this module for splitting and tokenizing text into parts with a reference to the instance also makes it possible to loop through the tokens afterwards looking for more information. With the help of regular expressions we can compare each token against expressions or words defined in code or in the ontology for match.

[11] NLTK - http://nltk.sourceforge.net

The tagger module tags the tokens with supplemental information. It uses regular expressions to find out whether it is a number or a string. Numbers are tagged with CD and strings are tagged with NN.

3.6 XBRL

3.6.1 Introduction

In the thesis definition the sentence "extracted knowledge should be captured in such a way that it can be exported into financial standards like" are given. One of the goals was to do express the knowledge found in the extraction by the standard XBRL. A brief introduction is made in this Section.

3.6.2 Definition and basics

XBRL is an abbreviation from the eXtensible Business Reporting Language. XBRL is designed to enhance the creation, exchange, and comparison of business reporting information. Business reporting includes financial statements, financial information, non-financial information and regulatory filings like annual and quarterly financial statements.

The working draft [XBRL03] contains definitions of XML elements and attributes that can (optional) or are to (obligatory) be used in business reporting. XBRL consists of a core language of XML elements and attributes used in instances of XBRL documents. The core language's abstract elements, which by definition not can be used as an instance, are replaced by concrete elements in XBRL instances. Abstract elements are defined in taxonomies. Taxonomies describe standard ways to distribute business information, and can be regarded as an extension of XML schema containing several hundreds of concepts. XBRL also consists of a language for defining new elements and taxonomies of elements.

One main goal of XBRL is to improve business reporting, following existing accounting and business domain standards. A standard format is to be used for storing the information, but different ways can be used to present it [XBRL03] [xbrl].

XBRL is extensible language enabling any adopter to increase its breadth of applicability, and is built like a hierarchical structure. A global high-level review is the foundation of XBRL to ensure that all major sections of primary financial statement are included. The next level is taxonomies created at country level. Here country dependent elements are added. In this way more elements are added for each level, like industry and company, disclosing specific elements [Ric02] [XBRL-FR]. Based on different compositions of these taxonomies used through namespaces, is XBRL instance documents made containing the actual financial data [Tib02]. All taxonomies are defined in XML Schema (XSD) files.

The taxonomies standardized at [XBRL-FR], are only the two most general ones. National extension together with industrial and company extensions are not provided as standards yet. Many other working draft taxonomies are also under development, i.e. at [xbrl].

XBRL allows faster evaluation of data as it separates information from style. XBRL instance documents are often constructed by other programs filling information and financial figures into the XBRL file, like accounting systems or other sources of information. Different examples of instance documents are; printed financials, regulatory filings, Web sites, tax returns and bank filings. The resulting XBRL file containing all the figures, are referred to as an XBRL instance document.

One detail of XBRL important to notice is that positive and negative numbers are differentiated by adding negative numbers with a minus sign. For example are both profit and loss placed in the same tag, like for instance "ProfitLossAfterTax"[12].

A tag called "numericContext" is used to describe the tags giving numbers, like in "ProfitLossAfterTax". An attribute in each element (e.g. "ProfitLossAfterTax") links to the id of the belonging "numericContext". Several elements of can link to the same "numericContext", as long as they have the same description.

In "numericContext" an important attribute named cwa is given. This attribute is a Boolean value indicating the validity of a "closed world assumption". If cwa= "true", then the reader of the XBRL document can assume that all relevant information are provided. When cwa= "false", we can not assume that all the relevant information is provided and calculating any new facts based on this can not be done [XBRL03].

In addition to cwa, are also units, like currency, precision and period some of the XBRL instances that can be defined in "numericContext". Some of these are optional. The complete list of instances can be found at [XBRL03].

3.6.3 Currency standard in XBRL

XBRL uses ISO standards to unambiguously express information where there is any chance for this. One frequently used, is ISO-4217 for describing which currency used to describe financial amounts. "$" and "€" are not used in this standard, but three letter notations like "USD" and "EUR".

[12] XBRL sample – http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15/Samples.htm

3.7 Current status of ontologies

Ontologies are a relative new concept in computer science. Since its introduction in the early 90's, several ontology languages have been developed as introduced in Section 3.4.3. OWL, as the newest ontology language, is still a working draft at W3.org and is only supported by a very few freeware ontology editors. RDF and DAML+OIL seem still to be used most. Most editors supports RDF, but many editors also supports DAML+OIL. A group of ontology editors are introduced in Section 3.3.3. My comprehension is, to make a summarization of current status, that the different tools all have advantages and disadvantages compared to each other. Differences are mainly according to; ease of installing; user interface; guidelines and tutorials; ontology language support; plugin support; and update frequency.

The freely available editors are mainly made at universities and are therefore more educational and not a "professional" editor which OntoEdit, and other applications from the On-to-knowledge project, seems to be. OntoEdit is not a freeware and license needs to be bought.

Searching the Web for projects guided or based on ontologies has not resulted in many hits relevant for the area of using it to extraction of information. Using this technique is quite new, where papers mostly are written after 1998-1999. Many of these only describing what ontology is and some have a small test case assigned. Bigger projects are often not published in detail. The two most interesting ones are described in Section 4.2.1 and 4.2.2. These are both a collection of separate projects which are put together for a complete working set of applications. Ontological differences are looked at in the project referred to in Section 4.2.3.

Except giving me some useful ideas and providing more theoretical insight in the domain of ontologies, the projects found has not given me many practical solutions. A conceptual solution has therefore been made, described in chapter 5. In chapter 6 the solution based on the conceptual solution is presented.

4 Ontology used for extracting information

4.1 Introduction

A preliminary study of current projects and status of extraction of data guided by ontologies has been done. The best projects, and most relevant work in this area to this thesis, are described in this Section.

4.2 Related work

4.2.1 The Artequakt project

The Artequakt [Ala03] project by Harith Alani et al. at the University of Southampton is an ongoing project with a publication in IEEE in the January and February edition this year. "*Automatic ontology-based knowledge extraction from Web documents*" is the title of the publication. The project is exemplified by biographies of artists. It automatically extracts information about artists from the Web and populates a knowledge base. The knowledge base allows queries for information. This project has drawn expertise from three separate projects where different areas have been stated.

The first area includes construction ontology, and automatically populating this from online documents given the ontology's representation and WordNet[13] lexicons to expand the list of terms in the ontology. The information has been implemented in Protégé and stored as a knowledge base (KB), the second key area. The third area has constructed narrative tools to query the KB for relevant facts through an ontology-server.

For knowledge extraction information extraction (IE) systems have been used to reduce the documents of natural language to tabular structures. These structures are used to retrieve fragments of documents as answers to queries. To avoid the use of templates, the IE processes are providing with direct access to concepts and relations in the ontology. Articles are first broken into paragraphs and then into sentences. To achieve this are different tools used in the extraction process. GATE and Apple Pie together with WordNet has been used to analyze syntactically and semantically the sentences for identifying relevant information to be extracted. Trusted sites have been defined and used as basis for quality control between query and result.

Information after extraction is provided as an XML file per document. Different biography templates are authored in a tool named Fundamental Open Hypermedia Model (FOHM), and defines templates ranging from database queries to KB queries where sentences must be constructed.

[13] WordNet – http://www.cogsci.princeton.edu/~wn

The Artequakt project seems to have overcome many problems. In their further development section, they state some important challenges. Semi-automatic ontology population is a difficult part, though it reaches a great quantity of data, the precision and recall may decrease. Managing persistent information, duplicates and overlapping information from different sources are some issues that need to be dealt with.

Co references are also referred to as a significant challenge. Breaking the documents into sentences may loose some references, like "*he*" is used in a sentence referring to the name of a person given in a previous sentence.

4.2.2 The On-to-knowledge project

The On-to-knowledge project[14] [Fen02] is a major Information Society Technologies project concerning knowledge management using an ontology-based approach, lasting from 1999 to 2002. An article in IEEE in November 2002 titled "Ontology-based knowledge management" gives a brief description of their work. Using the term knowledge management denotes the process of acquiring, maintaining and accessing information. The projects overall goal was to make a huge amount of electronically information more accessible by using ontologies to make searches more intelligent and not keyword based. Intelligent searches answers questions, while keyword based only matches words.

The approach used in this project includes three different levels. The lowest level performs extraction of information from structured and semi-structured documents, the last also called weakly structured information sources. An intermediate level contains an annotated data repository where automatic access to meta-data is provided. At the highest level both clients and providers can explore and modify knowledge domains through implemented push and pull techniques.

Along with the project, a group of applications have been developed. A brief introduction to some of them is following. Ontology construction is provided in a semi-automatic way by OntoExtract (from unstructured sources) and OntoWrapper (from structured sources). The OntoExtract tool, executes some of the same tasks to be solved in this thesis. It parses, tokenizes and analyzes the text, and generates nodes and relations between them in DAML+OIL ontology language. The final result is submitted to the RDFS repository server Sesame. Sesame uses the query language RQL to query these incoming RDFs and provide result for the high level tools. This includes a number of tools, among these OntoEdit where you can browse and modify an ontology. OntoShare supports and encourages for sharing of information. This works together with the projects work to enable personalization and ontology-based user profiles.

[14] On-to-knowledge – http://www.ontoknowledge.org

This seems to be the project which has reached furthest in the area of ontology-based extraction. Many tools have been developed, a book is written and some publications are made. But since they have come so far, tools are not freeware, and publications do not describe in detail what they do. OntoEdit can be downloaded as a limited edition, allowing 50 concepts, instances and relations.

Several case studies are done and solutions are provided for big companies, like Swiss Life and BT, and includes; Applications for large intranets providing fast and reliable access large documents; managing skills and job functions; disseminating of large collections of rules and regulations; and to improve knowledge transfer between in-house researchers and outside specialists via Web pages have been made.

In the next Section a project using ontology to solve the problem of different use of scales, formats and units in financial systems.

4.2.3 Knowledge integration to overcome ontological heterogeneity

A student project [Fir02] at MIT, USA, has developed an extension to the context interchange (COIN) framework for representing and reasoning ontological heterogeneities. Semantic interoperability among traditional and Web data sources are achieved. Different financial databases reporting the same companies are found to often have differences. No scale, unit or format is enforced; cases like 50 million or 50.000.000, equal to the issue I have in my case, has therefore needed to be dealt with.

Ontological heterogeneity, also called ontological conflicts, is defined as data items calculated differently. Profit is in some sources considered as the result before tax, and after tax in others. In the extended COIN framework a shared ontology is allowed to assume these different meanings from different contexts. A meta-ontology layer has been developed in order to do this assumption. A semantic type identifier is used to determine which type to use based on its context. This brings, according to [Fir02], flexibility into the system. The ontology is able to deal automatically with ontological conflicts; no changes in the ontology are required. A mediator performs a task of selecting sources that can satisfy a given query, which are financial figures.

This project reveals, in financial information systems, that data-level, ontological and temporal conflict is quite common. Global interoperability is depending on solving these conflicts. Ontology level components are illustrated in this project, and, as a part of the logic-based ECOIN (Extended COIN) framework, providing a solution of data-level and ontological conflict.

In the next Section is a conceptual solution, exemplified with an optimal scenario, for ontology guided financial information extraction presented.

5 Conceptual solution for ontology guided extraction

5.1 Introduction

For financial information to give any value for the user, they need at least some certain data. In the shortest and easiest form, we can say that we need to know what happened and when did this happen. If a company had some income, we are interested in when this happened and where this income came from, like sales of a special product. Further we want to know how much this income was, specified in some amount of currency, and which company this apply for.

Finding information is not always an easy job. One problem may be to find date and year, often occurring several times in an article and pointing to different periods or dates. Examples of this are when an action, like payment, took place; profit of some period ending at a certain date or the date the article was written. Another problem is all the possibilities of writing a date, which can be done in several different formats as both numerical and textual. Date and year may not always be stated in the article at all, assumptions, described in Section 5.2, have therefore been stated.

Another problem is, also described in [Ala03], where a pronoun, like "*he*", refers to a subject given in a sentence before. This may also occur in the financial articles about company, and period or year. For instance without knowing when the article was written, it can be impossible to find what year "*next year*" refers to. This problem is known as anaphora, [Cri01].

In the next Section assumptions for input file will be stated, a conceptual solution of the prototype will be presented in Section 5.3, followed by an optimal scenario in Section 5.4 picturing and describing a desirable result of the extraction process.

5.2 Assumptions for input article files

The anaphora problem can be quite difficult to solve when extracting information from the articles, if not any date and year this article was published are stated for example just below the title. "First quarter" and "annual report" are example of expressions used frequently used and can cause problems. However since information about the articles found by Intermedium's agent are stored in a database, also date and year found for each article are stored. Assuming that the article is gathered by the agent the day it is published, is it easy to store the article release date and year along with the text in the database. Anaphoric references can then be interpreted and expressions like "first quarter" can provide additional information about when this data is valid for.

The agent also knows where the article is found; this is the source. URL to the article is also provided. Having this information ready, will ease the extraction process a little regarding to setting time to the information.

The input to the system is assumed be an XML file where date, source, URL and the text to be extracted are tagged. Making a query engine to get the right information from the database and return it as XML is not a difficult task, neither a focus in this thesis. It will however be a pretty important part of the system which the prototype will be relied upon. In addition to provide important information about when this news refers to, it also provides a standard way of representing the date which can be used when storing output knowledge in standard format.

5.3 Using ontology in financial information extraction

5.3.1 XML input to extracting system

The *input* to the process will, as assumed in Section 5.2.2, be an XML file containing at least release date and year, and the text where the information to extract is. The extraction process is visualized by figure 7, where *Input* is presented at the left.

5.3.2 The extracting system, the agent

The retrieval of financial information from *input* will be carried out by *agent* who extracts information from the text variable. The application prototype is called an *agent*, and will provide a service that combines the process of text extraction with the use of ontology and to provide an output. To read the ontology, the *agent* will have to use an API or parser which supports the language and standard that the ontology is written in. A processing language or toolkit is needed for enabling the agent to understand semi-structured information. Without this, the agent will not be able to read and understand anything of the text given. In the human world it would be like we should read Chinese without having any knowledge of this language at all. It would all seem “greek” to us.

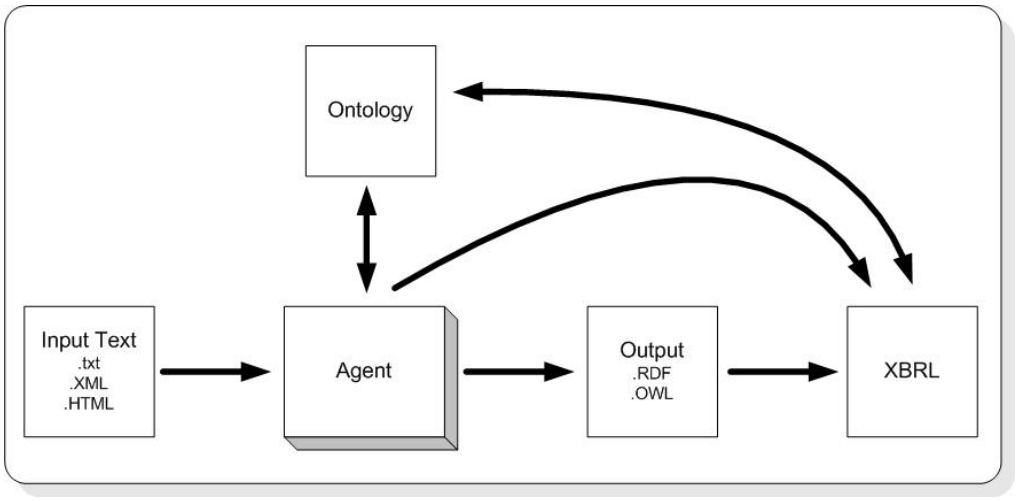

Figure 7 – Conceptual process of ontology guided information extraction

One main challenge in this thesis is how to get the extraction and parsing the *ontology* for right information in the right order, and get these two to work together for a better extraction process. Would it better to extract the text first and then check if every word exists in the ontology? Or is it better to parse the ontology and keep all words and instances in memory and then extract these words from the text? What's better may depend on the size of the ontology compared to the average size of the texts to be extracted. If the ontology is big, it will be more time consuming the more reading that has to be performed. An easier and better way will probably be an interaction where the *agent* queries either the ontology or the text when needed in the process. This will be especially true when a property of a resource is found and this new resource has a reference to another resource, which again is relevant only if the first resource is found. Here it is necessary to query the ontology again, unless the whole ontology is available in a searchable memory. Making a loop here, will give us an easy extensible agent where you will not need to enter the agent code, but only the ontology editor to complement your extracting system. An algorithm has to be made to clarify this issue.

5.3.3 Ontology

The idea is to build up a sort of net, or grid, of information in the *ontology*. The *ontology* will be a knowledge base of words which relates to each other as synonyms (words with approximately equal meaning, like income and revenue), relations (words that are in some kind of relation to another word, like income often refers to an amount of some currency) and antonyms (words with opposite meaning, like gain vs. loss). Using this technique, you know that if a currency is found, you want to know what it describes, like income or loss of something. The resource containing the unit of currency has a reference to, among others, the resource where income is denoted. The same applies to revenue, which practically has the same meaning as income and a synonym of the same concept. Further the resource, where income and revenue are defined, may have a reference to

another resource, like a resource describing what this income was a result of. This could for instance be income from operations, which several occurrences of have seen in articles provided by Intermedium's Web agent.

One important goal of this extraction process is that an improvement of the system will be done by extending the *ontology*, adding more words, relations or mapping to other ontologies, and not by extending or rewriting the code in the *agent*. Rewriting of code is a more difficult process and can be more time demanding than simply adding more effort into the *ontology*. This effort can be done automatic, semi-automatic or manual by humans as introduced in Section 3.3.4. New words, meanings or relations may be wanted in the *ontology* for and therefore enforcing expansion of the ontology anyway. The agent will not be finished by this thesis, but can at some time be considered done, while the ontology may evolve more or less continuously.

5.3.4 Output of extraction process to an ontology language

When the information extracting is done, together with use of the *ontology*, the output will be ready to be written to an external file *output*. An ontology language, like RDF or OWL is to be used to enable extracted information with semantic tagging and relation between them. Querying the result can be performed since the result will be in a computer understandable language.

5.3.5 From output to XBRL

Information extracted can either be transformed from the ontology language to the financial standard *XBRL*, or be made directly from the *agent*. Expressing the result of the information extraction in this format will provide the possibilities of using the result in all applications and by all business supporting XBRL. Probably will the amount of available software supporting XBRL increase with the time, as the standard grows and gains support. The customers have then the opportunity to choose which software that provides the best result for them. Maybe they already have a tool for exchanging financial information in XBRL, and then can use this application for querying or displaying XBRL information resulting in no extra cost buying new software.

Filling the XBRL instance document from the *output* in ontology language with information found by the agent can, as shown in figure 7, can be done by using an XSL file to read the output and map to the taxonomy given by XBRL. In the other solution, where the *agent* directly writes the XRL document, can a similar approach as for *output* be used.

The connection between *XBRL* and the *ontology* points to the desire of making the finance ontology as close up to and equal to the XBRL standard as possible. The more these two correspond, the easier it will be to map from the *output* to *XBRL*. It may also provide easier reverse engineering when, or if, lacks in the *XBRL* file are found that is appropriate to express in the *ontology*.

Why not give the result directly in *XBRL*? The thought is that it may be easier to use the *ontology* in the process of extracting information when thinking of some corresponding holes to be filled. Consider given example: One financial term is given, income. Together with income more facts are needed, like how much is the income in amount and currency. Further you need to know what this income is about, which could be from operations. In addition is date and year along with the source of this news already given. To summarized, we have now "income operations $5 million" and a good indication of when this was, based on when the news was written. This can be extended to include more information, like getting which quarter or year, company and department this is about.

5.4 Visualizing an optimal scenario

5.4.1 Overview

Input text is given as a string of sentences from an article, together with date, year and source information. An information extraction based on a financial ontology will be performed by the *agent* and *output* will be generated in RDF. An XBRL file will also be generated, like given in figure 7. To be able to see a clearer picture, will an example be given describing what to do.

Starting point of information extracting is defined to be currencies found in a sentence. Other starting points could have been chosen, because there might be given much financial information without saying anything about currencies. Percentage changes in shares and "gain in income" are just some simple examples of this. An alternative is a defined set of financial terms given as starting points, which needs to be found for checking more words in sentence for matches. This should therefore be easy to change. In this thesis, currencies are taken as starting point.

Using the NLTK toolkit for tokenizing articles into sentences allows each sentence from an article to be the source for an optimal extraction process. Given an optimal extractor and a complete ontology, what information will be extracted and given as output?

The sentence is "*The loss before tax is expected to be approximately GBP39 million.*" is given as input in figure 8. A sentence containing much information is chosen, not all of them do so.

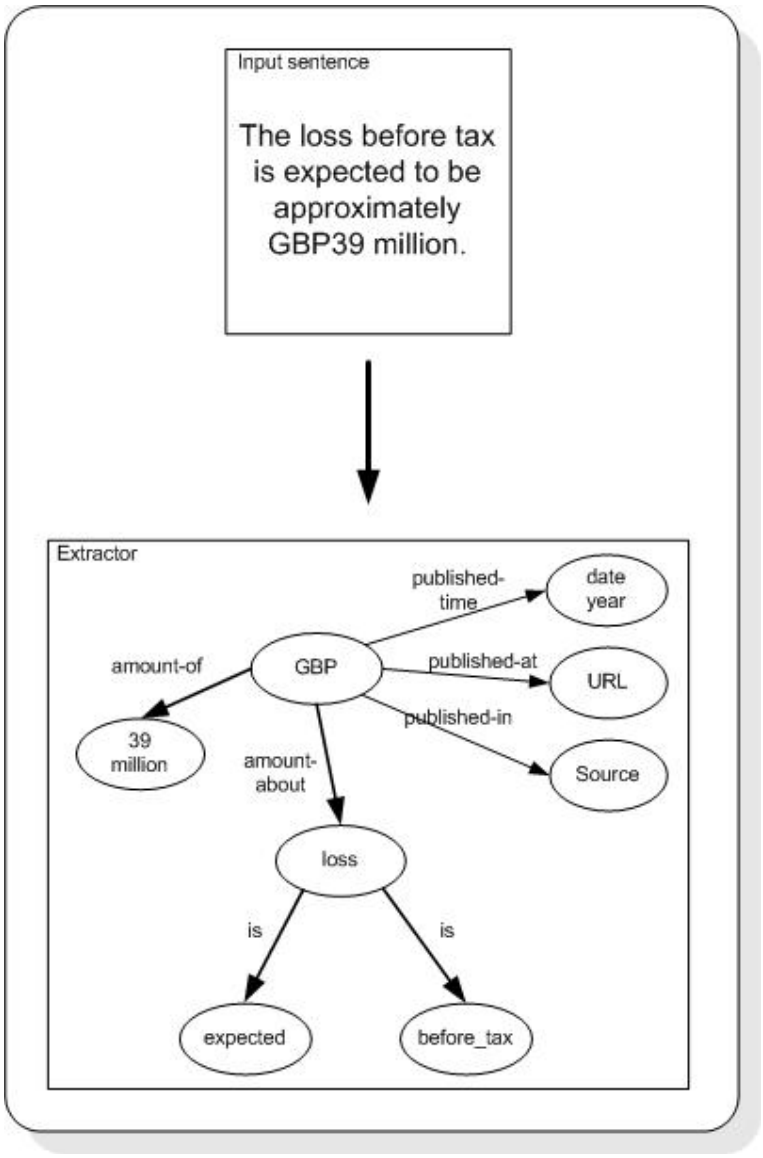

Figure 8 – Optimal extraction of example sentence

The extractor will first find *GBP* and then start a search for more information describing this currency. *39million* is found as the amount of currency in this sentence. *Loss* is found as a financial word, but this has modifiers like *expected* (and *approximately* both describing that this is not a final result) and *before tax*. All these words are defined in the optimal ontology, and there possible to locate and later assign meaning through semantics. The other words are just to complete the sentence in the form of natural language, and of no interest to extract. Date, year and URL are variables given at article level, and describe when and where this sentence was found. If a complete article was presented in this scenario, links from this to date, year, URL and source would have been made. These variables are not added in the sentence level, and are therefore not considered further in this scenario.

How can this extracted information be presented? There are many different possibilities, also within the ontology languages RDF and OWL. They both permit some individual differences as a result of the fact that to persons can make different ontologies about the same domain. Different ways of writing RDF/XML syntax are also allowed [W3C99rdf]. OWL is the latest working draft of ontology languages; but since RDF is described in Section 3.2.3 and this is a well defined standard providing all the functions which seems needed in this extraction process. In the next two Sections an example of an optimal output solution of this sentence in RDF syntax is considered.

5.4.2 RDF statement graph

A transformation from the extracted words in an unspecified structure, figure 8, to well defined structure given by RDF statement graph is presented in figure 9. The ontology is used to find meaning of each word found. URIs are chosen arbitrarily, as introduced in Section 3.2.3, ovals are resources while rectangles are instances and arcs are properties of the resources. SentenceID/1 is given this sentence, having an amount of currency and other words. CurrencyAmount is however considered as the starting point for extracting financial information and is in this example *GBP* and *39million*. It is also connected to profitLoss through routs-to, describing what information this financial amount is about. ProfitLoss are connected to modify and financeTax, it could have been more resources, but in this case it is instanced by *expected* and *before_tax*.

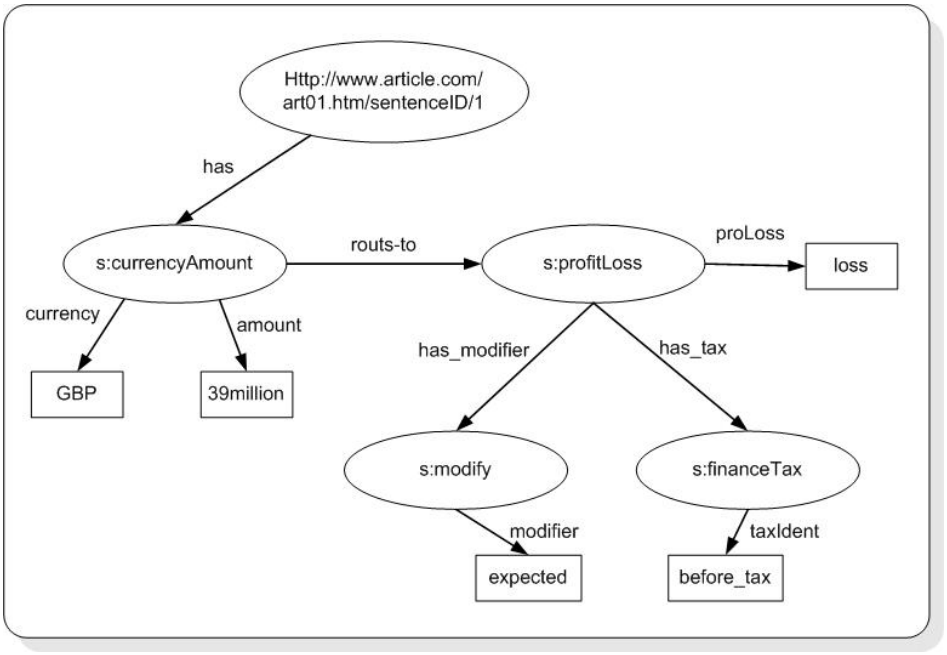

Figure 9 – RDF statement graph of example sentence

When the relationships are drawn, like in figure 9, in a hierarchical manner, it is clear that profitLoss are described further by modify and financeTax, and that *expected* and *before_tax* are used to provide more meaning about the profitLoss word *loss*. One alternative is that every oval could have been connected from sentenceID1. All the words would then have been at the same level where no internal relations between the words were made, only that they belong to the same sentence.

A sentence can contain several items of currency and amount. Then more new nodes would have been drawn from the sentence node in the same way as in the figure, where each bough is describing and belongs to the currencyAmount couple. Some properties can also be equal and count for more than one currencyAmount node. Two or more nodes can for instance refer to same period node.

5.4.3 RDF/XML syntax

Representing the information given in the graph, figure 9, in format of RDF/XML, readable and processable for machines, can be made in some different ways, as previously mentioned. Following the rules of RDF/XML syntax, will the programs be able to read and understand this syntax, like example given in table 2.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.ebjoraa.no/art01.htm/sentenceID1">
    <s:has>
      <rdf:Description about="http://www.ebjoraa.no/currencyAmount">
        <s:currency>GBP</s:currency>
        <s:amount>39million</s:amount>
        <s:routs_to>
         <rdf:Description about="http://www.ebjoraa.no/profitLoss">
          <s:proLoss>loss</s:proLoss>
          <s:has_modifier>
           <rdf:Description about=" http://www.ebjoraa.no/modify">
             <s:modifier>expected</s:modifier>
           </rdf:Description>
          </s:has_modifier>
          <s:has_tax>
           <rdf:Description about="http://www.ebjoraa.no/financeTax">
             <s:taxIdent>before_tax</s:taxIdent >
           </rdf:Description>
          </s:has_tax>
         </rdf:Description>
        </s:routs_to>
      </rdf:Description>
    </s:has>
  </rdf:Description>
</rdf:RDF>
```

Table 2 – Example in RDF/XML syntax

The arcs from figure 9 are assigned the namespace "<s:" defined in RDF Schema file. These connect the resources, like currencyAmount to profitLoss. Several triplets of information are given in this example, like subject *financeTax*, property (predicate) *taxIdent* and object *before_tax*.

The example sentence in natural language has been broken up into words and, guided by the ontology, given semantic meaning expressed in RDF/XML syntax. Transforming this result to XBRL may be performed by XSL or another application, or directly from the agent. XBRL will give the information with an accurate and uniform description of the meaning, and provide easy exchange of data between application and users supporting this language.

When the words and numbers given in natural language in a sentence are given well defined tags, is the information seen as knowledge. It can be queried and used by other applications to present the information for the end user in a well arranged way. The user will no longer be forced to read through the entire article to catch the information given.

An optimal scenario has been presented in this Section as RDF graph and in RDF/XML syntax. Construction of the ontology guiding the extraction process will be described in chapter 6. Based on figure 7, chapter 6 is divided into five main parts describing; input file, agent, ontology, output, XBRL, in addition to an introduction Section.

6 Ontology guided information extraction prototype

6.1 Introduction

Developing a prototype for information extraction guided by an ontology require several different applications, standards and programming language with additional libraries where all these are working together. In the process several different applications and libraries are found which support needed tasks. The construction of ontologies is one part where several different editors could have been chosen, another is libraries for reading ontologies. All the choices have been taken either based on recommendation from the supervisors or as a result of testing between alternatives and choosing what at that time seemed best suited for wanted purpose.

In this chapter an overview of the entire process is presented, starting with input articles in XML, financial ontology and application prototype, to output in RDF and XBRL. The sections describing the financial ontology (6.3) and the agent prototype (6.4) are both main parts of this thesis definition and a greater part of the programming performed to solve the questions given. They are placed under this chapter, although they also contain some discussions and are described in greater detail then the other sections. This is chosen in order to be able to provide a full description of the solutions and why it is solved in this way. To understand how the agent work, a description of how the ontology is constructed has to be presented. And to see how the outputs are being made, an understanding of the ontology structure and methods used by agent has to be presented.

6.2 XML input to be extracted

6.2.1 Article information as input XML

The articles are stored in a data warehouse at Intermedium, where each business that subscribe this service, are using its own data mart for querying for information. How the agent work are not given any focus in this thesis.

Information is gathered and stored in the database by the agent. In addition to text, information about source like URL and language the article is written in, together with date and year found, are stored. Querying a database is an easy task and therefore not been given any focus in this thesis either. The agent is based on input from XML files giving the information needed, also given as a condition in Section 5.3. The actual text from the article, date published and the source (URL) are used in this prototype. A complete XML file example is presented, where all used input variables listed above, are presented in Appendix J.

6.3 The financial ontology

6.3.1 Introduction

An existing ontology for the financial domain has been searched for, to be used in prototype, without any luck. None free ontology where located and found suited for the purpose of this work. One online financial ontology[15], or taxonomy, was recognized. A request for a free copy of their knowledge base was however rejected. Free online ontologies seemed either to be small example case or not appropriate for the financial domain.

It was therefore constructed a new ontology from scratch, which can be a time-demanding process. Making a complete ontology require much work and knowledge, as described in Section 3.3, a small version of an ontology for the financial domain which demonstrates its possibilities in this area of use has been developed. A preliminary study of 50 articles has been made as a background for the ontology construction; this is described in the next Section.

6.3.2 Preliminary studies of articles

To figure out what information needed to extract from the text, a study on a collection of randomly selected articles found by the agent for two companies has been performed. 50 articles were gathered, 25 from each company's site at Intermedium. They were selected from different sources and examined for information of financial interest. Currencies in text are defined as starting point of this thesis. When this is located, information about the amount must be found and then as much information describing this as possible, or wanted, to figure out its meaning. I therefore concentrated on finding information surrounding monetary measures.

While studying these articles, notes were made about what word that where used to describe the currencies. Among these words, many of them gave different information. There were also many words practically saying the same thing, like for instance income and revenue. Words that mean the same thing are known as synonyms. Words meaning the opposite of others words, like profit and loss, where also found. These are referred to as antonyms.

Going through a limited set of articles for developing an ontology financial domain, gave a good background for the further work of constructing the ontology. Many articles are written by the same publisher, like Reuters and Forbes. Often companies have policies for writing articles in layout of articles, and it seems so in writing them to. This especially counts when the same author writes articles. Writing a report about the annual result does

[15] InvestorWords - http://www.investorwords.com

not give so many options telling the same story either. This is however not the case when the report is given in table form, which are not dealt with in this thesis. The result of this is that the mostly used expressions can be covered in a limited amount of word.

Words to get started were located. Online dictionaries, freeware and taxonomies have been used for finding even more synonyms providing the ontology with a higher hit rate. WordNet, a free downloadable dictionary developed at Princeton University, is one these which is very helpful. Another one I will emphasize is Bartleby's[16] online dictionaries. However Bartleby's dictionaries are more time-consuming and also not available from any extracting applications.

WordNet is a powerful little program for browsing in a net of words and relations registered in a database. The relations in the database include, among others, descriptions of the searched word, its synonyms, antonyms, hyponyms, and conditional terms. By using these two dictionaries, the net represented in the ontology were enlarged. Testing against different articles has also resulted in finding some new words, added were into the ontology. Construction of the ontology is described in the next Section.

6.3.3 Constructing the ontology

Constructing an ontology is a demanding process, ref. Section 3.3.2. Although you have some of the entries which is to be stored here, you have to consider how to build it up. Relationships and dependencies have to be modeled in a meaningful way and for the purpose of use. In this thesis it is to extract information from semi-structured text files.

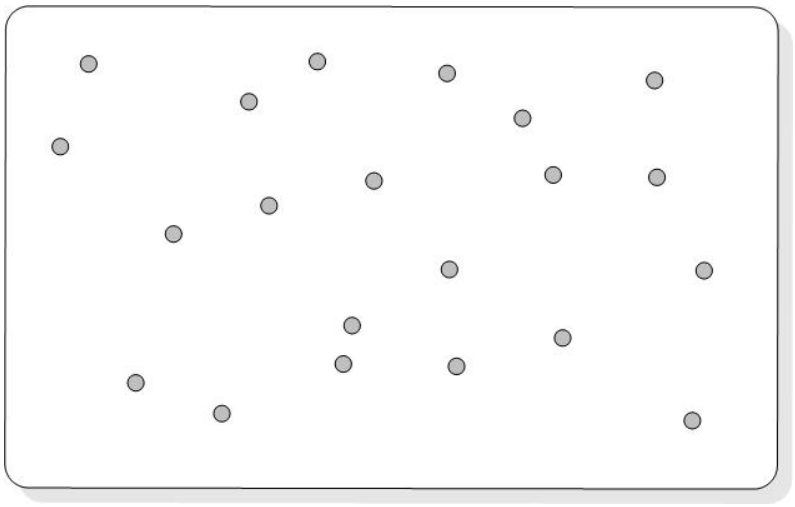

Figure 10 – Words without any given relations

[16] Bartleby's online dictionary - http://www.bartleby.com/62/

Consider first the words found by reading the articles, described in Section 6.3.4, before putting any relation about synonyms or antonyms into the ontology. This case visualized could look like figure 10, where the circles represent words found in the articles, or concepts as described in Section 3.3.1. Adding synonyms to each word will increase the resilience of the concept and the chances of it being captured in the extraction process. No relations have been stated, so no connection between the words can yet be made, see figure 10 compared to figure 11 and 3.

A sentence containing information can be seen as a series of words or concepts relating to each other. To ease the extraction process a little, a prune the words not needed. Because, when extraction information, there is no need to know every word. The important ones are numbers, subjects and verbs. The other words may only be there to formulate the sentence right in natural language.

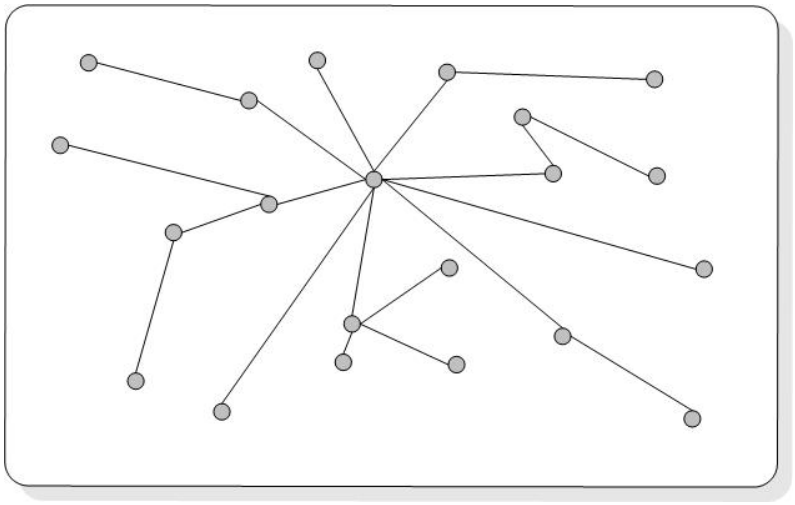

Figure 11 – Words represented in an ontology with relations

The starting point of this financial extraction was to find whether or not a sentence in an article contained financial figure; a currency and a number giving the amount. A concept containing different currencies of interest have been defined; in the ontology called *currencyUnit*. This is shown in figure 11 where the currency concept, *currencyUnit*, is placed in center. The *agent*, from chapter 5, will read the ontology for the starting point. Currencies defined will be found and listed; like ”€”, “$”, “USD” and “NOK”. If one or more of these is found in the sentence, the agent will find the amount and the relation to other concepts *currencyUnit* have. After this it will find synonyms for every of these concept and check the current sentence for match. If a match is found, a new check for relations to other resources is performed for more words and synonyms.

Having relations between concepts, see figure 11, gives the opportunity to search for a word, or a group of synonyms, only on a given set of conditions. When a word given in concept A is found, then, and only then, a search for words given in the relating concept are performed B. The advantage of this is that the search and work of the agent are limited. The extractor is not interested in a word unless a certain condition is fulfilled, and it is therefore not necessary to search for all concepts for a match in the sentence. The agent will, as a result of this, be able to work faster.

Alternative entrances can be made for extracting information. A defined set of concepts are defined as start point, a concept can be a blank node, see Section 3.2.3, only referring to other concepts where all these synonyms will be checked for matches and relations to other concepts. The advantage is that information from sentences that does not contain currencies can be captured. The downside is that the agent will get more work to do, more words to check for match and therefore be a little slower.

6.3.4 Ontology editor: Protégé

From the tools available on Web, some of them described in Section 3.4.5, are Protégé chosen for the ontology construction. Protégé is easy to install, easy to use and have good documentation. In addition a very active mailing list is available, and frequently updates and new builds are available. Protégé is also used by the Artequakt project, Section 4.2.1. Protégé 2000, version 1.8 build 1064 was selected.

6.3.5 Naming in Protégé

There have been many contributors to the evolution of ontology, including those that have made applications for constructing and editing ontologies. More then one name has therefore been used referring to the same thing. One of these is concepts, which are constructed as classes in many applications, including Protégé. Classes can, as in object oriented programming (OO), be built as a hierarchy of classes and subclasses. An instance of a class is an individual representation of the class, like in OO. An instance of a subclass is also an instance of the super-class. *CurrencyUnit* and *ProfitLoss*, seen at figure 12, are instances in the financial ontology.

In Protégé properties are called slots, a name used in the rest of this chapter. Synonyms (“$”, “NOK”, “€” and “krone”) and refInstance (URI reference to another concept) are examples of slots in figure 12.

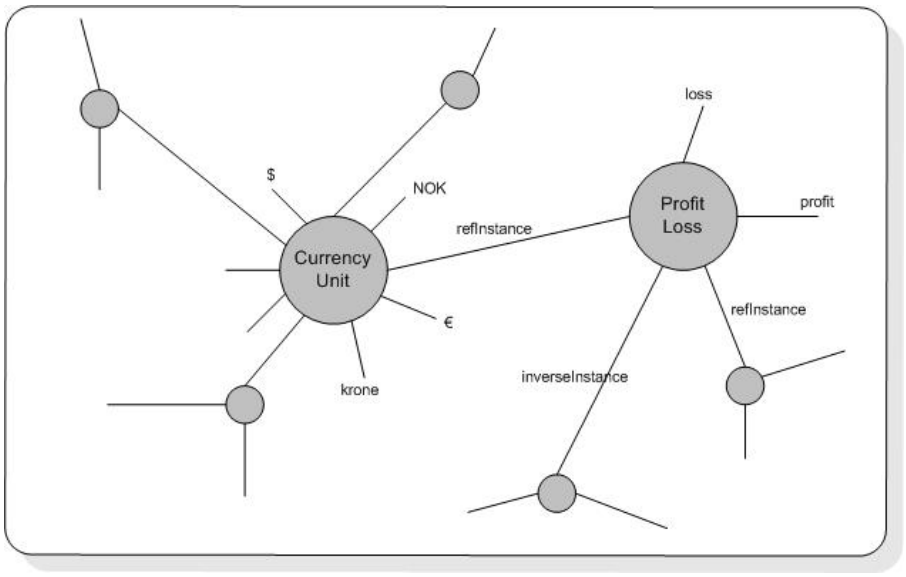

Figure 12 - Concepts in the ontology

A quick summarize before going further; the ontology describes the domain of finance with different classes (concepts); slots (properties) describes various features and attributes of the classes; in addition there is restrictions on the slots, also called role restrictions. The ontology together with a set of individual instances of the classes constitutes a knowledge base [Pro101].

6.3.6 Financial domain ontology configuration

When dealing with extraction of information from semi-structured text, it is important to remember that there is no single way, or any global pattern, for writing information in natural language. Many combinations of words can be put together and end up as a sentence where the meaning can be almost the same. To capture as much useful information from the text as possible, a way definition of interesting information has to be made. In this thesis this is done by the ontology. The more fine-meshed the net for finding information are, the more precision and recall of information will be provided. There is however a limit here. If it is too fine meshed, it may be difficult to decide what the information means, like example in Section 3.3.1.

Synonyms and relations between instances are the two main elements in the used ontology to store information; they are both designed as slots and a little sample of this was introduced in figure 12. Synonyms may not always be the right word for it; similarity could have been used as well. Sometimes it is also used for opposites, like profit and loss. This particular case is made because in many XBRL schemas define special tags, like the one called “ProfitLossBeforeTax”. Profit and loss are both in the same tag, separated by positive or negative numbers. This is described more in Section 6.6 about XBRL output. The slot synonym is however used to group information which has about the same

meaning or in some way are related and defined in a concept; currencies ("$", "€") are another example of this.

Relations, called refInstance, are used to define relationships between the instances in the ontology. This works as follows; if a synonym of an instance is found in the text, the agent should check for references to another instance. If one or more references are given, these instances are also checked for synonyms against the text, otherwise does the agent not need to search for this information in the text. This applies from the beginning of the extraction process; if no amount of currencies is given, no extraction will be done. If, for example, "$50" is found, the agent starts out with locating references of currency and then finds its synonyms.

6.3.7 Standards in financial ontology

Under the development of the ontology, different standards relevant to financial information has been used as much as possible. UNeDocs', United Nations electronic Trade Documents, code for currency, and in some degree country codes, are two standards used. The XBRL specification has also been considered. ISO-4217 has been used to support XBRL's standard for currency. A look into Schema files used by XBRL one can see that XBRL gives very precise definitions in the tags used, e.g. where both profit and loss are defined in the same element tag. An example of this is already given in Section 6.3.6 by "ProfitLossBeforeTax". This is described more in Section 6.6.

6.3.8 Financial domain ontology details

The ontology is divided into three super classes; *ReplaceText*, *MeasurementUnit* and *Finance*. Several sub-classes and instances, with relations defined as refInstance between them, are made. Some other slots, in addition to synonyms, have been added for describing the instances. InverseInstance has been added between some of the instances denoting a group of inverse words. The XBRL standard uses definitions of tags, meta-data for data presented, describing both profit and loss in the same tag only separated with positive or negative numbers or signs. The instances where profit and loss are synonyms could have been inverse instances. This is skipped because of XBRL's definitions of tag. XBRL is described more in Section 6.6. For the complete ontology in RDF/XML format, see Appendix A and B.

TextReplace

TextReplace is a class containing no sub-classes, but have some instances. It is used before the agent search and extracts information from the text and its purpose is to ease the further process of locating and extracting information. For example can $50 million also be given as $50,000,000. A standard way is supported by replacing every million and 000,000 to M (for Million), like $50M. Billion (B) and thousand (T) is also replaced

in the same way. The abbreviations, like M for million, are adopted from articles found by the news agent at Intermedium, used by magazines like CNN Money[17].

Currencies, like "$" and "€" are also changed to USD and EUR, following UNeDocs and ISO-4217 standard.

While developing the agent, some cases have been discovered that can easier be done in the ontology. One example of words being transformed from two into one is "first quarter" which is replaced by "first_quarter". This is done because the NLTK tokenization of text, introduced in Section 3.4 and described more in Section 6.4.4, where a comparison between each single word in the text and synonyms given in the ontology (as long as a currency occurs) are performed.

Measurement

MeasurementUnit is a class containing different measurable units, containing two sub-classes; *TimeUnit* and *AmountUnit*. The first refers to, as the name indicate, instances describing different units of time. Four instances of this class has been made; period which among others hold "first_quarter"; month where each month is given; year where the years from 2000 to 2004 are given; and period_info describing when a certain action happened, result occurred or a further description of a *TimeUnit*; like before and ending.

AmountUnit have a central place in this extraction application. As previous mentioned, is the extraction process triggered when the extractor discovers a currency in the article of current interest. The instance CurrencyUnit holds information about the different currencies of interest, like "€", "USD", "$" and "krone". Several of the registered currencies are gathered from UNeDocs currency code list, where each currency is given a three letter notation. There is however few of these used in the studied articles. Therefore are also signs like "$" and "€"; and some country dependent currencies like "krone" which is valid in Norway, Denmark and Sweden; added as currencies of interest. The currency signs "€" and "$" are in the agent prototype replaced with "eur" and "usd" to ensure consistency to XBRL and ISO-4217. They are however assigned lower-case letters of the same instance in the ISO standard, and the user is able to see that a transformation has been done from the original source. This applies only to the three-letter description of currencies following ISO-4217, and not currencies like "krone" and "cent".

[17] CNN Money article – http://money.cnn.com/2003/05/02/news/companies/epicor_st_jude.reut/index.htm

Finance

Class *Finance* contains several sub-classes, some of them also containing another level of sub-classes. The name *Finance* is chosen because of this is the collection of different classes and instances containing words often related to or found in the finance area. The two most important sub-classes of *Finance* are *ProfitLoss* and *Prediction*. *ProfitLoss* contain a group of instances dividing financial terms like loss to *financeNegative*, profit, buy to *financeInput* and sell to *financeOutput*. Several other instances and synonyms to the ones given as examples here are registered in the ontology, seen in Appendix A.

Relations between instances have been tried developed to provide as global pattern of expressions. If a synonym of an instance has been found, only then instances of another instance will be checked. For example if a synonym, like loss in *ProfitLoss* is found, instance *measurePositive* is checked for synonyms like gain or increase and *period* for finding i.e. which quarter the loss apply for.

Synonyms and relations for the instances in the ontology are based on the gathered material found by reading in all over 50 articles from Intermedium's agent, supplemented by some more synonyms from WordNet and Bartleby's dictionary.

It has not been a goal to make an "complete" ontology for the financial domain making all information thinkable about this domain stored. This would have required more domain skills and more focus on the ontology construction part. The purpose was to, based on a set of terms represented from a representative set of articles, to find out how the ontology could guide the extraction of knowledge to work better. To increase the effort the ontology provides, the main thing is to add more instances, synonyms and relations.

An prototype application, called agent, has been developed to extract information. The agent extracts information based words defined in the ontology. In the next Section this is described.

6.4 The Agent; financial data extractor

6.4.1 Introduction

The agent, introduced in conceptual solution Section 5.3.2, is extracting information from articles of XML format based on words defined in the ontology. This process is looked into the next Section, while reading and querying the ontology follows in Section 6.4.3. In the preceding Sections of chapter 6.4; natural language processing; the extracting algorithm; classes in the agent; problems under development; and some further extensions are described.

6.4.2 Parsing input XML

The articles to be extracted are, as described in Section 6.2.1, given in XML format. To parse and get the information from the different tags, an XML parser supported in Java™ 2 SDK, Standard Edition, called "javax.xml.parsers" has been used. Javax parser builds a nodelist based on a Document Object Model (DOM). Several functions are defined for DOM's nodelist; two of these, getElementsByTagName and getNodeValue, are used in the java file getTextFromOnt.java to parse the XML file, see figure 13. A node is the set of a start and end tag with value in between, while a nodelist is a list of all the nodes in a document. By matching getElementsByTagName against the nodelist and using getNodeValue, the values in the nodes are retrieved.

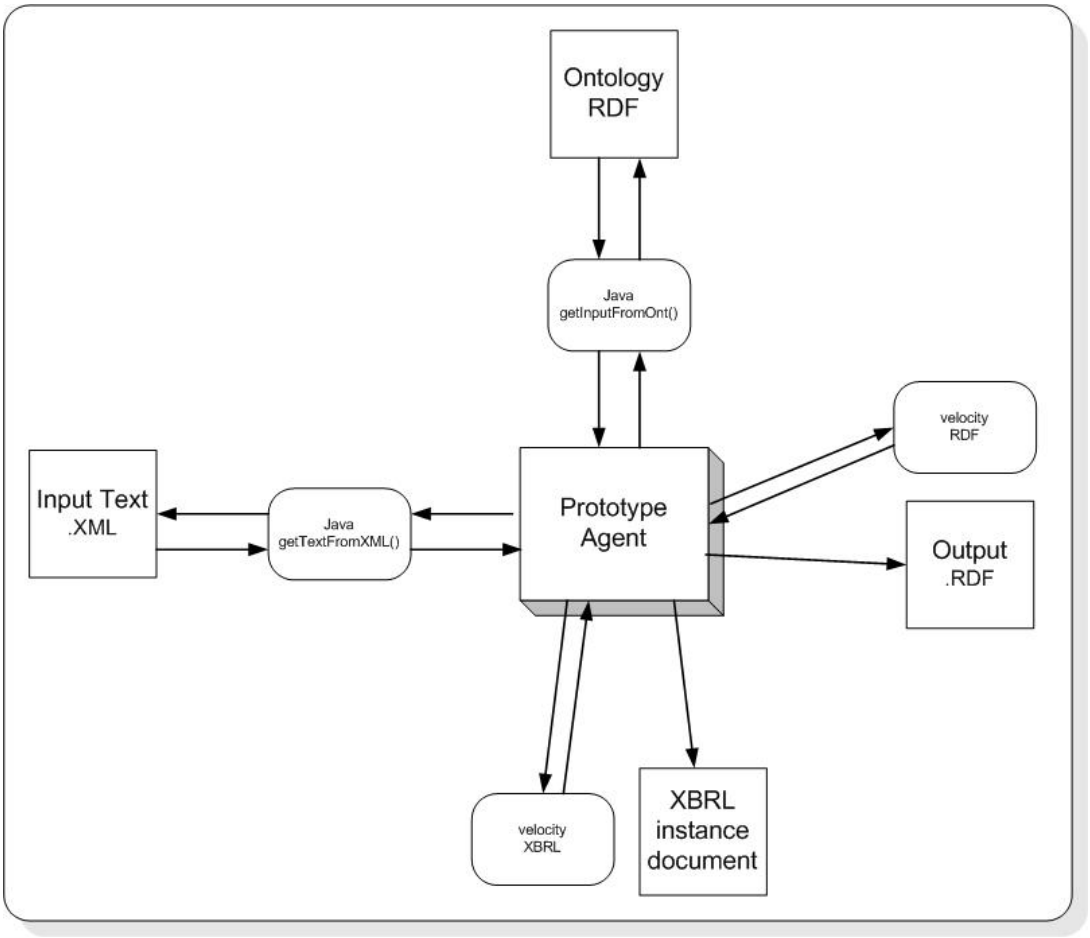

Figure 13 – Agent prototype architecture

What file and which node to search for information, is given as parameters into the function. This ensures reuse possibilities, and the function is used to get input information from all the tags. Usually when dealing with an input files containing many nodes to get values from are repeatedly calling of the same function not any good solution. An alternative solution with for instance a vector of all nodes searched for, and

a vector with all the result returned, could have been made. However, since there are only a few limited nodes given in the input has this case not been prioritized. The source code to getTextFromOnt.java can be seen under Appendix D.

6.4.3 Parsing and querying the ontology

RDQL, introduced in Section 3.4.6, has been found most suitable to query the RDF ontology. The query is performed in getInputFromOnt.java, see figure 13.

The function for finding slots to instances in the ontology is built as general as possible, where parameters are specifying the file, slot and output queried for. The query for finding relationships to other instances from *currencyUnit* is one example of the queries made to the ontology. The query is shown in table 3. The answer of this query will be several URL's to different instances *currencyUnit* has a refInstance to. All these are placed in a vector and returned to the calling function.

```
SELECT ?x,?y,?z
      WHERE (<http://protege.hia.no/ebjoraa/kb#CurrencyUnit>,
             <Finance:refInstance>, ?z)
      USING Finance FOR http://protege.hia.no/ebjoraa/kb#
```

Table 3 – RDQL query example

"SELECT ?x, ?y, ?z" can be given a arbitrarily name as they are used as temporary variables in the query. Including "?x" and "?y" are not necessary in this example since only "?z" is used. The "WHERE" part defines in which RDF triple a match is looked for. The first part of the "WHERE" part is describing which instance queried for, given by an URL. "<Finance:refInstance>" denotes that the next constraint specified is the slot refInstance of the instance. "?z" is used to store the result of the query before returning it to the calling function. The last line is used to maintain a more readable query, without the long URIs obscuring the structure of the pattern [hp03]. The complete java code for querying the ontology is listed in Appendix E.

6.4.4 Natural Language Processing in the agent

Natural Language Processing (NLP) and the toolkit NTLK was introduced in Section 3.4. NTLK is basically a collection of classes and function, which are added into the Lib catalog of the Jython directory. Making use of NLTK is done through regular imports and function calls.

Two of the properties of NLTK have been extensively used by the agent, namely tokenizing (token.py) and tagging (tagger.py). Some adjustments have been done to make these two functions better serve the agent's needs.

In tokenizing of the text from articles, the function LineTokenizing has been modified from tokenizing line based to sentence based. A sentence end is here defined as either a dot with a following white space (". ") or a question mark ("?"). This will not tokenize 100 percent right, since writing and dot errors may occur in articles. Errors like this are not dealt with in this tokenizing process.

Tagging the text for deciding whether it is a number or a word, has also been added some regular expressions for additional allowed instances of numbers. This is done to cover as many different variations of writing currencies as possible, like $60,9 and GBP98M. M (for million and 000,000) is used to provide a consistent presentation of financial figures, described in Section 6.3.8 and 6.4.5, and this transformation is performed before tagging of words. Numbers and financial figures are tagged "CD" and words are tagged "NN".

The functions in NLTK where the changes have been done are listed at Appendix G. For further description and the rest of the code in the toolkit NLTK's Web pages[18] are referred.

6.4.5 Algorithm for extracting financial information

In figure 14 is a high perspective algorithm written as pseudo code presented. The algorithm describes the most important functions, loops and if tests the agent performs while doing financial information extraction. This includes reading the input file, query the ontology, extracting information form the text and printing the result to RDF and XBRL.

Each line in the algorithm, except before begin and end, is given a character reference, like "a)". This is used to easier see when loops and tests are finished. It is also used in the next Section, 6.4.6, where the agent's classes and functions are described. The functions are given the reference to which task they solve in the algorithm.

[18] NLTK - http://nltk.sourceforge.net

```
a) get article info, add to dict
b) do text replacements for better extraction
c) do find all currencies in ontology
d) tokenize article into sentences
e) for each sentence do:
    begin
f)    do tokenize sentence into words
g)    for each word in sentence, do tag word with CD/NN
      begin
h)      for each word, do check if it is a number (CD in tag)
          begin
i)           if CD found, do check for currency
               begin
j)               do handle $ (change to usd) and € (to eur) differently
k)               if currency found, do look for synonyms/financial expr
                   begin
l)                    add currency and amount to tempDict
m)                    get all refInstances from currencies in a vector
n)                    if refInstances already used for synonyms do skip
                        avoid same synonym twice
o)                    for each instance in vector in m), find it's
                        synonyms
                      begin
p)                      get all synonyms for instance in vector
q)                      for each synonym, do check against every
                          word in sentence for match
                        begin
r)                        if match and instance has refInstance do:
                            begin
s)                             append word found to sentence list
t)                             check for new refInstances
u)                              if new unused refInstance found do:
                                  begin
v)                                  add new references to vector
                                      in o)
                                  end t)
                              end r)
                        end q)
                    end o)
w)                  update list of sentence with tempDict from l)
                  end k)
               end i)
          end h)
      end g)
    end e)
x) update dict with sentence list from v)
y) print dict to RDF output file
z) print dict to XBRL output file
```

Figure 14 – Algorithm in pseudo code for the extraction prototype

6.4.6 Classes in agent

The agent is the kernel of this extraction part, see figure 13. The ontology is an important part too, but it is the agent that makes use of all the different inputs and tries to find the information specified. It is also responsible for the output. In this Section is the prototype agent developed, and how it works for extracting information, described. References, like (a), to corresponding action in the algorithm, figure 14, are added to the functions where these apply. Some lines in the algorithm have more than one reference given below. This is because the algorithm has been abbreviated, in addition to some functions which calls other functions for task performed. Where this is the case, both function are referencing to the algorithm.

The agent is coded in Jython, which is Python programming language with the extension of being able to import and use Java libraries which python do not have. Java functions can therefore be used to retrieve input from the articles and the ontology.

Template, *InstanceList*, *TextInOut*, *TextChange* and *TextExtract* are the five classes that have been made to separate the different functions developed in the agent.

Template()

Velocity Template Language (VTL)[19], a powerful scripting language, also used for example in dynamic Web sites, has been used to print the output. Providing this template file with variables to parse and text to print, have a complete structure for output in RDF and partial for XBRL been made. These templates are described in Section 6.5 and 6.6. Input variables for Velocity are given as a dictionary. How to parse this is defined in the *Template* class. A "Python to Java" transformation of the dictionary to array lists and hash tables are defined, and placed under the template() class. This class has been developed at Intermedium.

The dictionary in Jython is filled with all the information extracted from the financial article. The first elements in the dictionary are article dependent variables, like subject, language and URL, and apply for all sentences. The "sentences" list contains all sentences in the article, and all information extracted from the sentences. Examples of this are currencies (CU), amount (CA) and the synonyms found based on the ontology. Each sentence can contain several currencyUnits, therefore can the list "CuCa" contain several dictionaries where each contains a currency and amount couple. An example dictionary is given at table 5.

[19] VTL – http://jackarta .apache.org/velocity

The dictionary is sent to the velocity files for parsing. This Velocity files contains a defined template for printing the given data. One is made for RDF and one for XBRL, see figure 13. Example article variables, like subject and language, are also given in this dictionary. The template is returned and printed to file. For complete VTL code, see Appendix H.

```
{'Subject': 'Financial result',
 'Language': 'English',
 'Source': 'Reuters',
 'fileName': 'xml/inputSentence.xml',
 'pubDate': '20030515',
 'myRDF_file': 'Finance.rdf',
 'URL': 'http://www.yahoo.com/finance/art001.htm',
 'sentences':
 [
  {
   'ProfitLoss': 'loss',
   'tax': 'after_tax',
   'CuCa': [{'CA': '39M.', 'CU': 'GBP'}],
   'predict': 'expected',
   'financeNegative': 'loss',
   'SText': 'The loss after_tax is expected to be approximately GBP39M.
    ::',
   'SID': '1'
  }
 ]
}
```

Table 5 – Example dictionary to Velocity file

InstanceList()

InstanceList (n,u) is a class of functions for making, adding, reading and deleting a list of instances which at the time has been checked for synonyms. A new list is made for every sentence. This is to prevent never ending loops. It also contains the dictionary, and a function for updating this which is parsed by the velocity file.

TextInOut()

TextInOut contains three functions where the first two functions are a bridge to the java class files getTextFromXML.java and getInputFromOnt.java. GetTextFromInputXML() (a) gets strings of text, URL and all the other information given in the article input file. The other function, getWordsFromOnt() (c,m,o,p,t), gets the information from the ontology by calling the getSynonyms from the function getInputFromOnt in the java file. The result is returned as vector. The last function, printToFile() (y,z), prints the result from RDF and XBRL Velocity templates to output files.

TextChange()

TextChange contains three different functions for making changes in the text. TextReplace() (b) replaces words and dots (“.”) that does not mark the end of a sentence with “;”. This makes it easier for the lineTokenizer from NLTK, see Section 6.4.4, to break the article into sentences. Regular expressions are used to perform this change.

Another text replacement is ”\n”, which in Jython is the textual denotation of a new line. This is replaced with a white space. Otherwise “\n” would have been given in the result if a match word was follow by this sign.

TextReplaceFromOnt() (b) are much the same as TextReplace(), only getting words to remove and replace from the ontology. This is done for two reasons. The first is to ensure consistency of equal terms represented differently. For instance is the English “billion” and Norwegian “milliard” both replaced by “B”. The other reason is to make it easier to group and find two, or more, words belonging to each other, like “first quarter” which is replaced with “first_quarter”. This is because of tagging into single words that are compared to the synonyms from the ontology.

MyTagger is the demo() function copied from tagger.py of NLTK, which calls the tagging function in tagger.py, which has been adjusted as described in Section 6.4.4.

TextExtract()

This final class contains five functions responsible for matching and extracting data from the text. FindNumbInSentence() is the first function called when starting to look for financial figures in the text. First tokenizing (d) the article into sentences, then sentences into words (f), and thereby tagging with CD or NN (g). All currencies in ontology have been selected into a vector (c). All these tasks are performed by calling other functions. For each sentence, every word is checked for numbers, tagged with CD. All numbers are check for currency assigned. “$” (Dollar) and “€” (Euro) signs are treated with special if statements; “$” because it is a sign used in regular expressions as the symbol meaning “at the end” of for instance a sentence; “€” because of it has provided big programming difficulties due to text encoding. This problem is described further in the Section 6.4.7. Another way around these problems has been added. “$” and “€” are replaced with “usd” and “eur” (j). Then also consistency against ISO-4217 and XBRL has been secured, but the change is still visible to the user. When currency and amount is found (i), the function findWithRefSyn() is called.

FindWithRefSyn() performs a check to find out if an instance, given as an in parameter from findNumbInSentence(), have references to other instances (m). The references are found by calling findRefInstances() and returned as a vector. If this vector contains any references, one by one is sent to findSyn() (p,q) to find synonyms and perform information extraction. If a synonym is found in findSyn(), findRefInstances() are called.

A check for references to other instances is performed (r). If this is true, a vector is returned from findSyn() back to findWithRefSyn(). The vector in findWithRefSyn() is updated with the new references returned (v, updating vector in p). If not a synonym is found, you will not look for references to other instances, even if this is given for the current instance. This loop is never ending; there can be unlimited levels of hierarchy. The only limitation is that two instances are not checked for synonyms twice, directed by findRefInstances().

FindSyn() finds all synonyms for current instance and loops through all words in sentence for a match with given synonyms (p,q). If a match is found, the list of sentences in the dictionary is updated (s,w). This list, the same as described above in this Section, are at the end sent to Velocity. If the instance has reference to other instances, refInstances is returned as a vector back to findWithRefSyn() as just described.

FindRefInstances() (t) finds all references in the slot refInstances, uses checkList() from the class InstanceList() to check whether or not this instance already has been searched for synonyms (u). A list of all the refInstances is returned as a vector. The complete prototype code is listed in Appendix F.

Some programming problems have occurred, in the next Section the one most troubling are described and how it was resolved.

6.4.7 The "Euro" problem

Euro is a currency used by many nations in Europe and necessary to include in the ontology as one of the currencies that should be searched for in the articles. Problems with different encodings of the input files and automatic encoding of text in Jython, has provided many hours of testing and figuring out how to solve this problem. Asle has provided useful help during this process.

The problem starts with Protégé storing RDF/XML in the only supported encoding format "ISO-8859-1", also called latin-1. Handling of the currency unit Euro, "€", is not supported in this format. "ISO-8859-15" is one encoding that do support "€". In the beginning, this was used as encoding for input files. Comparing Euro from two different encodings does naturally not work. The input file was therefore changed to "ISO-8859-1" using an explicit definition to support Euro. Definition in XML is provided like this; <!DOCTYPE Text[<!ENTITY euro "€">]>. All Euro signs from the article is assumed to be expressed as "€" or "€".

Euro is expressed as "`€`" in the ontology and no manual change are needed to be done. In the agent an if test is checking if the currency is "`€`". When this is true, the variable is changed to "unichr(0x20ac)", "IS0-8859-15" encoded Euro. This needs to be done, because "€" from the input text has automatically been transformed into this

char code. The code "#8364" is in ISO-8859-1 used for euro (€) and understood by Jython as "unichr(0x20ac)". Then a check of this against the currency found in the text is performed.

6.4.8 Extensions of the agent

A Python interface to the WordNet, a database of word meanings and relationships, has been found and tested with some simple command line queries. Download files and examples are found at footnote[20]. The interface enables queries against the stored relations in the database; like synonyms, antonyms, hyponyms, and hyponyms.

It has not been implemented in the prototype, but it could have been used in a semi-automatic way of expanding the list of terms in the ontology, like in the Artequakt project in Section 4.2.1. Using this interface, queries for finding synonym and other relations to other words, would have provided an extended list of terms in the ontology. This could reduce work in construction of the ontology, or provide more precision and recall than not using the WordNet database. A manual check for consistency and correctness is probably to prefer for avoiding errors in the extraction process.

6.4.9 Agent summary

The agent and its surroundings, functions for ontology querying and reading XML input file, have been made as global as possible. The purpose of this, is that making the extractor better mainly are supposed to be performed in the ontology by creating new instances, relations and adding synonyms. If major changes however are wanted, changes in the agent code are also necessary.

In Section 6.5 the result of the extraction process and output in RDF format is described.

6.5 Extraction results in RDF

6.5.1 Introduction

From an article where text is given as natural language the extraction process, guided by ontology, search for financial information of interest. Semantics can be added to the financial figures and expressions as they are found thanks to the ontology describing words and relations.

Adding semantic to information can be done in many languages; XML, RDF, DAML+OIL and OWL for instance. RDF has been used throughout the entire thesis, from describing the Semantic Web, conceptual solution and construction of ontology in Protégé. The output has been tried to look as much like the conceptual solution, the

[20] Python interface to WordNet – http://pywordnet.sourceforge.net

optimal scenario given in Section 5.4, as possible based on the information found and programming of the agent. Output in RDF is chosen. RDF enables query of information and a transformation from the output format to XBRL, as suggested in the conceptual solution, are possible.
The financial ontology contains, as described in Section 3.2.3, of two files; RDF and RDF Schema (RDFS). The financial ontology in RDF is used by the agent to find financial information in the given text. It is also a foundation to produce the output in RDF in a way looking something like the financial ontology. Structure in the output could then be built based on the ontology and more dynamically then with an explicit structure defined.

There is however one difference from the financial ontology RDF to the output RDF that do not allow reuse of the RDFS file, and forces another approach. Articles are, as described in the Section 6.4 and 6.5.2, broken into sentences before extracting information. For each sentence the agent will look for concepts stated in the ontology, and match these against the words given in the sentence. The financial ontology is at the sentence level, used over and over again for each sentence for match; while the output RDF will contain many "information objects". Many sentences, but far from all, will provide us with information we seek for. All this information can be seen as "information objects", and an article can contain many of these.

Constructing a minor ontology for the domain of articles will provide the RDF output with a schema file. Information applying to every sentence or at article level can be conceived in a syntactical correct manner. This is described in the following Section.

6.5.2 Article ontology

The financial ontology does only provide definitions for words to extract for each sentence in an article. Information about the article stored at Intermedium's database, like URL and source, are not defined in the financial ontology. An additional ontology, called article ontology, has therefore been made to declare all these properties. This ontology is also constructed in Protégé, and its structure is shown in RDF statement graph syntax in figure 15, as introduced in Section 3.2.3. The literals found are gathered from the same sentence used in Section 5.4. For complete RDFS see Appendix C.

The extracted result in RDF/XML syntax is using both the financial and the article ontology schema as definition of the tags used. An example result of this is presented in Section 6.5.3.

The article resource, given by its URL, is the main or starting node for defining words and relations between the resources of interest. The article has four direct literals of information, in figure 15 given example values. Properties (arcs), given by #source, #pubDate (date of publication) in format like 04302003, #language and #subject.

An article contains of many sentences, in the figure two sentence resources are inserted. Each sentence is a part-of an article, given by the #part-of property of the article. The complete sentence text is given as a literal of the sentence. The text is always printed in the output file, also when no financial information is found in it.

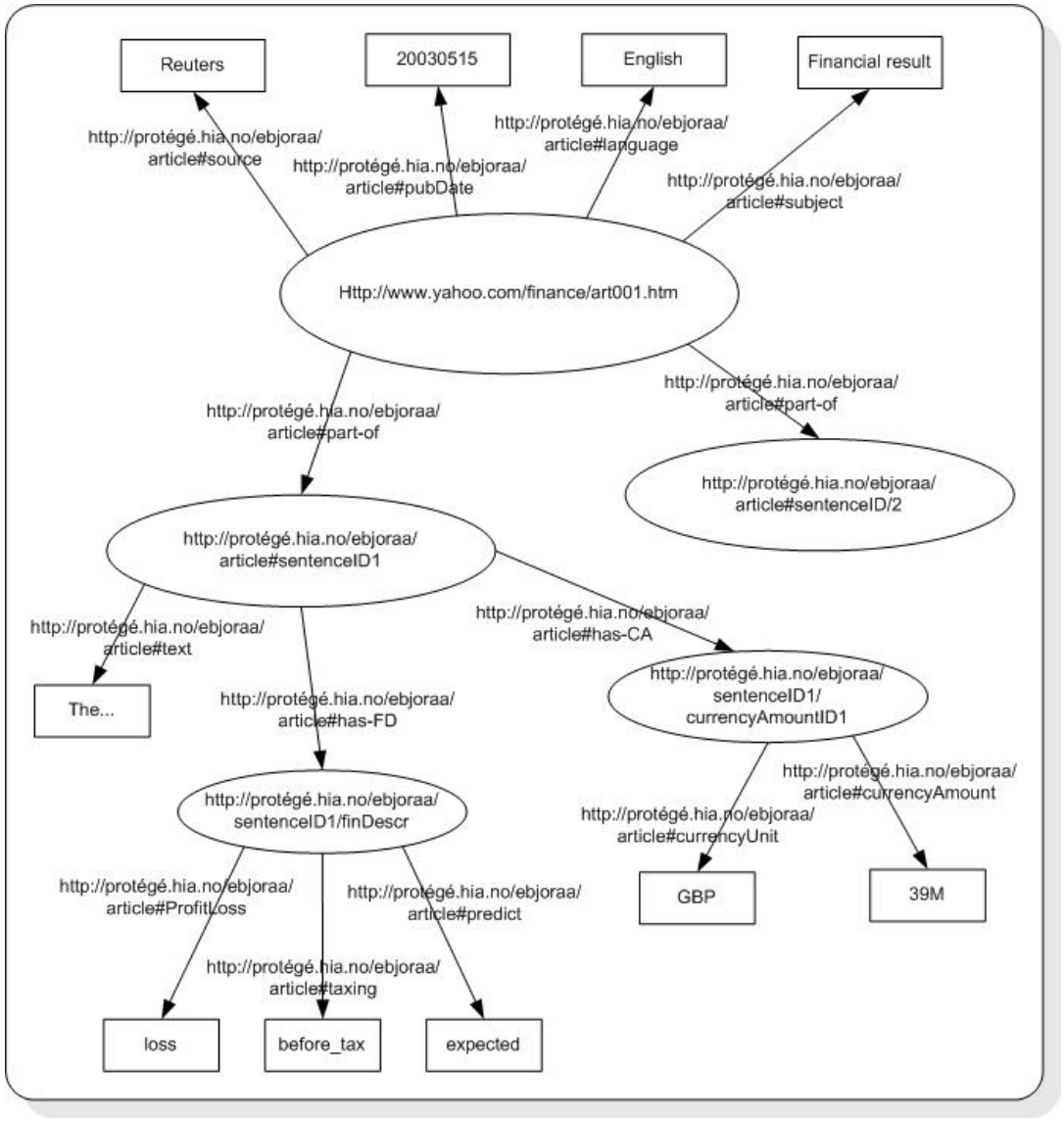

Figure 15 – RDF statement graph of the article ontology

The extractor starts searching for information looking for currency and amount. The resource #currencyAmountID is added when these are found. Currency and amount are coupled and can not be separated. A sentence can however contain more than one #currencyAmountID couple, and they are added like #currencyAmountID1 is in the figure. In the optimal solution, displayed in figure 9 Section 5.4.2, a property from the s:currencyAmount to s:ProfitLoss is made. This is optimal, but one problem dealing with

natural language, is finding what financial description belongs to what currency and amount. Different ways of referring meaning can be done, as the example below.

> "*Total and net revenues are anticipated to be broadly in line with expectations at approximately GBP31 million and GBP17 million respectively*."

Description of currency and amount are described by "respectively". In some sentences are information separated by comma, in other it comes in a long sequence. Finding what information belongs to which number, is not dealt with by the prototype agent. If the agent had been able to do this, changing of the structure of output, like given in figure 9, would have been easy to accomplish. Much works are however required to make the agent able to deal with this problem.

When the agent only finds one #currencyAmountID couple in a sentence, the assumption that all financial expressions most likely are used to refer to found #currencyAmountID resource can be made. One exception is the few times when information is referred to something given earlier or later in the article. This is not considered in the prototype. #currencyAmountID contains always both currencyUnit and currencyAmount.

#finDescr is the resource where all words found in sentence describing #currency-AmountID are placed. Compared to the optimal scenario in figure 9, are all the information given as literals directly from the #sentenceID resource. In figure 9 some words are used to describe others. Such a structure has in some degree been used in construction of the financial ontology, but not in such a degree that it is implemented in the output file. A deeper research into words belonging and describing another word further will discover this. Many words are used only when some other are used, while other can be used both alone and together with other. Getting this complete with high precision and recall of extracted information require a deeper study in this field. Focusing on sub areas of the extraction process have not been prioritized; rather completing a working sample.

6.5.3 Output in RDF/XML syntax

Templates made in Velocity have, as described in Section 6.4.5, been used to generate output files from the information found. The same example sentence as in figure 15 has been run through the extractor, resulting in the output file given in table 6. In addition to the information found in the sentence, is example article information also given.

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<!--  Output from textfile: inputxml/02.xml   -->
<!--  RDF file: Finance6.rdf  -->
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
<rdf:RDF xmlns:kb="http://protege.hia.no/ebjoraa/kb#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s=http://protege.hia.no/ebjoraa/article/sentence#
  xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
  <rdf:Description
   about="http://biz.yahoo.com/djus/030512/1653001230_1.html"
   s:published="2000512"
   s:language="English"
   s:source="Dow Jones Business News"
   s:subject="Corixa Results: 1Q Financial Table, Comment">
  <kb:part-of>
    <rdf:Description
     about="http://protege.hia.no/ebjoraa/article#sentenceID1"
     s:text="The loss before_tax is expected to be approximately
      GBP39M. ::">
     <s:has-CA>
       <rdf:Description about="http://protege.hia.no/
        ebjoraa/sentenceID1/currencyAmountID/1">
         <s:currencyUnit>GBP</s:currencyUnit>
         <s:currencyAmount>39M.</s:currencyAmount>
       </rdf:Description>
     </s:has-CA>
     <s:has-FD>
       <rdf:Description
         about="http://protege.hia.no/ebjoraa/sentenceID1/finDescr">
         <kb:predict>expected</kb:predict>
         <kb:ProfitLoss>loss</kb:ProfitLoss>
         <kb:tax>before_tax</kb:tax>
       </rdf:Description>
     </s:has-FD>
    </rdf:Description>
  </kb:part-of>
 </rdf:Description>
</rdf:RDF>
```

Table 6 – RDF output

An example article, independent from the articles from Intermedium's agent, is given in the Appendix K together with the result of the extraction process.

6.6 Extraction results in XBRL

6.6.1 Introduction

XBRL is the second output format of the extraction of financial information, introduced in Section 3.5. To produce an XBRL instance document, a similar approach as for writing RDF output has been chosen. The same dictionary is sent to a Velocity template adapted for XBRL, see figure 13. Much of the information found in the articles is for example predictions, and not real results. This has to be stated in the XBRL instance document.
The articles searched within have very rarely many financial results, the articles which do have tables of information, like a representation of annual result. These articles have not been considered for the extractor. The information written to the XBRL document can therefore be some insufficient. To solve this, different exceptions handling different missing information has to be made. The information found, must also be added to the right tags.

Several different taxonomies can be used to define tags in a XBRL instance document. A standardized schema for electronic financial reporting from IASCF[21] [XBRL-FR] has been used in an example for financial statements[22]. This has a limited amount of elements declared in the schema, and has been used in the XBRL part of this thesis.

To provide an working example of an XBRL instance document, an limited support of information found are dealt with. This is described in the next Section.

6.6.2 Output as an XBRL instance document

The same dictionary as used for making the output in RDF has also been used in the template for the XBRL document. The dictionary is previously described in Section 6.4.6 and 6.5.2.

The XBRL template, see Appendix I, has been constructed based on the example referred to in footnote 22. The element “ProfitLossBeforeTax” and “ProfitLossAfterTax” from the taxonomy schema are used to express currency and amount described by the instance “ProfitLoss” in the financial ontology. Examples of synonym words in this instance, is; profit, income, revenue and loss. A check for “before_tax” or “after_tax” specified in the text are done, and based on this the right tag is used. The amount of any currency found is displayed between the tags.

[21] IASCF – International Accounting Standards Board – http://www.iascf.com
[22] XBRL sample – http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15/Samples.htm

"numericContext" is used to define the tags of financial numbers with more information. URL identifier, currency (ISO-4217) used for the amount, measures of integers (e.g. decimal), end date, valid period and duration are all examples of this. Many of groups of "numericContext" can be defined. They are linked from the other elements, like "ProfitLossAfterTax", with an attribute referencing to the id of a "numericContext" group of tags, like "Currenct_forPeriod".

In table 7 an example of an XBRL instance document created from extracting the sentence "*The revenues are USD50,000 before tax for the fourth quarter*." are presented. A new example from the one given for RDF is presented to include the presentation of period informtion.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!--  Date/time article created: February 12, 4:31 pm ET -->
<group xmlns="http://www.xbrl.org/2001/instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:iascf-pfs="http://www.xbrl.org/taxonomy/int/fr/ias/ci/
   pfs/2002-11-15"
  xmlns:iso4217="http://www.xbrlSolutions.com/taxonomies/
   iso4217/2002-06-30"
  xsi:schemaLocation="http://www.xbrl.org/taxonomy/int/fr/ias/ci/
   pfs/2002-11-15ias-ci-pfs-2002-11-15.xsd
  http://www.xbrlSolutions.com/taxonomies/iso4217/2002-06-30
  http://www.xbrlsolutions.com/taxonomies/iso4217/2002-06-30/
   iso4217.xsd">

 <iascf-pfs:ProfitLossBeforeTax numericContext=
    "Current_ForPeriod">50T</iascf-pfs:ProfitLossBeforeTax>

 <numericContext id="Current_ForPeriod" precision="Not given"
   cwa="false">
   <entity>
     <identifier scheme="http://www.yahoo.com/test001.htm">
      Express Scripts, Inc.</identifier>
   </entity>
   <period>
      <duration>P1Q</duration>
      <endDate>2003 -12-31</endDate>
   </period>
   <unit>
     <measure>iso4217:USD</measure>
  </unit>
 </numericContext>
</group>
```

Table 7 – output from XBRL instance document

"ProfitLossBeforeTax" is found to be 50T, where T denotes thousand defined in Section 6.4.6. This tag has a reference to the "numericContext" with id "Current_ForPeriod". Here URL and source are, period and the measure defined. The duration is found in the text to be the fourth quarter, the year is not found in the sentence, but a good guess can be

made based on the date when the article was found. If the article is published in January, fourth quarter probably means the ending of the year before. If the article is published just before or in the fourth period, and in addition a word like expect or approximately are used, we can assume that this is a prediction of a result and the year should be the same as when the article was published. A limited if checks are added in this part of the template.

"cwa" is always set to false in the prototype template because an assumption that all the relevant information is provided can not be made. Therefore can not any calculating of new facts only based on this be done [XBRL03]. Precision is one of the optional XBRL instances that have not been specified in this prototype.

Numbers are in the extractor changed from for instance 50 million or "50,000,000" to 50M for consistency, as described in Section 6.4.6. In XBRL documents it is written as "50000000", so a change needs to be performed. This is an easy task in the extractor and is not dealt with in this prototype.

7 Discussion

7.1 Introduction

The thesis work started out with searching for projects using ontology for extracting information from online sources to investigate the current status in this area, and to get some ideas for the prototype to be built. As the extraction of information has been limited to financial figures and information; a search for existing ontology within the financial domain has been done. A proper ontology was not found, therefore a financial ontology was constructed used by a developed a prototype agent for extracting information. The information found in the semi-structured information sources have been transformed into both RDF/XML syntax and XBRL instance document.

The purpose of this work has been to see if ontology can guide the process of extracting information from semi-structured info sources. Based on the words and relations expressed in the ontology, the prototype should be able to know what to extract.

This discussion will include several major parts in this thesis structured in the order they have been dealt with. Issues discussed are:

- Current status in the area of ontology guided extraction.
- Ontology guided information extraction.
- Extracting financial information in RDF and XBRL includes a discussion of the formats used and how they are produced.
- Results of ontology guided extraction
- Further work that of interest in this area. .

7.2 Status on ontology guided extraction

A study for related work in the area of extraction of information based on an ontology has been done. The two best projects within the extraction area are referred in Section 4.2.1 and 4.2.2. Ontology is providing valuable help in their extraction processes. Both projects are a collection of projects, have lasted for several years and still have some work to be done, especially the Artequakt project. They are however using automatic methods for solving some issues, like ontology construction, which makes it more difficult to ensure validity.

In Section 4.2.3, a student project providing a solution for ontological differences from databases and Web data sources is presented. A solution of the problem when words are used about different terms, like in this case profit that denotes before tax in one source and after tax in another source, is provided. In this project figures are given and its

meaning defined, though differently in some cases. The problem in this thesis is that it might not be stated whether the profit is before or after tax.

In general it seems to be more and more interest around semantics Web and the use of ontology to help finding information. The search engine Google has recently acquired "Applied Semantics" to be able to do more advanced queries.

7.3 Ontology guided information extraction

A search for an existing financial ontology has been done without any luck, and a new ontology was therefore developed from scratch. Appropriate dictionaries have been used to find synonyms, antonyms etc and added manually into the ontology.

To construct the financial ontology Protégé has been chosen. The ontology language RDF has been used all along from the financial ontology, article ontology and to RDF output. Newer ontology language is available; for example do DAML+OIL and OWL support more relations between instances in the ontology than RDF. In this prototype RDF supports all the relations needed and RDF is easy to query and interact with.

Extracting information from semi-structured sentences is not easy for computers. Words or patterns and regular expressions defining what to extract, has to be made. One advantage of ontology is that you can define instances of words with mutual relations in a file, and use this as a guidance of what to extract. A prototype application has been developed performing the whole extraction process. Starting by reading the input file; extracting information based on queries of words and relations defined, in and returned from, the ontology; and at the end providing the result in RDF and XBRL.

When using ontology guided approach to extract information, an extension of words to be searched for and extracted will be done by extending the ontology. No change in the prototype will be necessary because the application handles instances and relations through querying the ontology. This is an essential part, though constructing ontologies are time-consuming, it requires less work than developing new or change extracting applications. The same application can also be used to extract different kinds of information based on the assigned ontology's domain or scope. The financial ontology could for instance be made in different languages.

Compared to the traditional way, extracting information by NLP use of ontology will ease the burden of the extracting application. Relations from one instance to another in the ontology are only explored when a match is found for the first instance. When no currency and amount is found in a sentence, which is the starting point of extraction in this prototype, no further word matching is necessary. Directing the extracting process can then be accomplished depending on the words found.

The application is, in the current version, not able to find out which financial words are used to describe the different figures, when more than one figure occur in a sentence. Many different ways of referring to a number in a sentence can be made when dealing with natural language in semi-structured sources. An extensive algorithm to deal with this has to be made. This could have been done by using NLTK's methods of chunking sentences and building a sentence structure. Some effort could also be made in the ontology. Defining frequently used words and patterns can help NLP to couple belonging information. This part has been excluded from this thesis, but is an important part of an end product for extracting information.

7.4 Extracting financial information into RDF and XBRL

The results of the extraction process are stored in a dictionary, described in Section 6.4.6. This dictionary is used by two Velocity templates for RDF and XBRL output. Velocity templates are easy to change and provide a clean way of printing the result. An alternative is to print the output when it is found. This would have resulted in a less tidy programming code, if even possible, due to its structure and relations.

Choosing an appropriate output format is important to enable further use of the information found. Computers should be able to process and query the information found in the extraction process. To enable this, the ontology language RDF was chosen. An article ontology has been constructed and its RDFS has been used to declare the tags of article information in the result RDF file. All tags are therefore valid and the result can easily be processed and queried by a computer application. RDF is also the language used in the financial ontology queried by RDQL. RDQL can also be used to query the result RDF.

XBRL is mainly used to publish and exchange complete financial reports, like annual results, between different users [xbrl]. All information about the financial numbers is available for the applications making the XBRL documents. When dealing with semi-structured information sources, you will not always be able to find all information about the numbers. It is therefore difficult to make it completely right according to the element tags and "numericContext". XBRL has a way of denoting that the information given may contain errors or lacks through the attribute "cwa", Section 3.6.

In the conceptual solution in Chapter 5, two different ways of making the XBRL instance document has been introduced. The first solution was to use an XSL file to parse the output into XBRL, the other to generate the XBLR document directly from the prototype agent. This approach was used because a dictionary in Jython had already been made making the work easier. This solution also seemed to be at least as good as the other in being able to express the figures in the right semantic way. The templates are also easy to edit, and making loops and perform queries for the values given is also straight forward.

Only a limited XBRL instance document has been implemented in this prototype, see Section 3.6. To make a complete XBRL instance document, the template has to be extended to place all the extracted information found in right XBRL elements. An example of how this is done for "ProfitLossBeforeTax" and "ProfitLossAfterTax" are given in the XBRL template.

Knowing which tag to use are defined in different XSD files. When the extractor search in a large amount of articles, a great variation of information is found. Finding the right XBRL taxonomy supporting the information extracted is important. It may be necessary to create a new or add missing elements in an own XBRL XSD to capture all information correctly. Some editors are available for this purpose, for instance at Fujitsu[23].

The extractor does not find out what financial numbers belong to which words. It is therefore not possible to deal with this in the XBRL part either. If the extractor knew which words belonged to which financial figure, it would not be difficult to add and support this in the XBRL template.

Many of the articles are not providing a lot of financial information. To gain more knowledge, a fusion or clustering of several XBRL documents having something in common, like company or period, could have been performed. A more complete picture of knowledge found can be provided, and a comparison of values found for correctness can also easily be made. This could have been solved by setting some constraints and let the agent search through a group of articles, placing all fitting result in a dictionary and send to XBRL template for output. Tools may also be available for comparing different XBRL documents, this has not been checked, but this will most likely be the best solution. The agent is then not forced to perform new extracting processes.

7.5 Results of ontology guided extraction

The result in RDF/XML syntax of the extraction process compared to an optimal solution, presented in chapter 5, are quite equal. One difference is however that in the optimal solution the words are describing financial results connected to the financial figures. The extractor is not able to find which numbers and words that belong to each other, and is a problem when more than one financial figure occurs. This problem has not, as mentioned above, been dealt with in this thesis.

Another difference is that all words extracted are placed under the node "finDescr" (figure 15), while in the optimal solution each word have its own node (figure 9). In the optimal solution this means that a word can define another word further. Since no relations between the words are found in the extractor, this can be hard to accomplish for

[23] Fujitsu XBRL taxonomy editor – http://www.labs.fujitsu.com/free/xbrlconv/en/xbrltaxedit.html

the extractor. It can however be solved by using the relations in the financial ontology. Taking care of the relation between the words extracted only because another word was found, can enable this. This case has not been considered as an important part of this prototype nor as a big challenges to solve, and therefore left to further work.

7.6 Further work

There are several issues in this thesis that could be a subject for further work. Some cases have during the development not been considered or prioritized. Brief presentations of some interesting areas for further work are given below.

- Taking care of relations between the instances in the ontology based on which synonym found and words extracted, should be considered. This is however not seen as a major task.

- The agent has not been developed to be able to differentiate which financial figure words are referring to when more than one figure are given in a sentence. Another student thesis at HiA is dealing with a problem similar to this and perhaps their solution can be integrated in this prototype. NLP and NLTK helped by ontology specifying words used to refer to figure, like “respectively”, can be used to analyze sentence structures, and based on these find dependencies between words. The artequakt project has also used Gate and Apple Pie to analyze the sentences semantically and syntactically. Their approach should also be considered solving this issue.

- In this thesis a minor financial ontology have been made. An effort to make this bigger would improve the knowledge extraction. A semi-automatically approach, almost like in [Ala03], populating the ontology with words from WordNet could be used. A manual check of the ontology afterwards will most likely be necessary.

- The XBRL instance document is far from complete. Only an introduction to how this can be done is given in this thesis. An XSD handling all financial figures of interest that can be found need to be found or made. Then the template should be extended to being able to deal with all information found, and placing it in the right tag with right reference to “numericContext”.

- Other and newer ontology language provides more relations. Lacks in RDF has not been discovered in this prototype, but newer languages supports more relations and properties added which might come useful in an extension of the prototype. If a more professional application is to be made, buying license to editors and tools, used in the On-to-knowledge project in Section 4.2.2, should be considered.

8 Conclusion

Retrieving computer processable information from texts in natural language has mainly been done by using natural language processing and specified by regular expressions. The main target of this project has been to extract financial information from semi-structured documents, like Web pages, and capture this in a way that provides more knowledge by giving the information semantic tagging. A financial ontology has been constructed and an evaluation has been performed of how an ontology can provide a better extraction process and more knowledge in the result by developing a prototype.

The thesis started out by searching for related work in the area of information extraction guided by ontology. Three projects ([Ala03], [Fen02] and [Fir02]) have been described in Section 4. The first project use ontology to build up a knowledge base of artists; the second have developed solutions, based on ontology, for better retrieving information from e.g. large documents. While the last uses ontology to handle different use of terms in financial systems.

Example Web articles found by an agent at Intermedium has been used as real world examples. These have been investigated to find financial words used. This work was the background for constructing a financial ontology, which have been done in Protégé editor with the ontology language RDF. Additional words and synonyms have been added by the help of online and downloadable dictionaries, e.g. WordNet. The structure of the ontology has been to group synonyms, or words used to express similar thing, in the same instance. Also some opposites have been put in the same instance. Income and loss are example of this. This is because this is used in the same tag in XBRL to express information.

A prototype application has been made for extracting information. A query against the ontology is made for words to look for in the text. The extraction is done by natural language processing and regular expression matching of words from the ontology and the text from the article. A dictionary (Jython) is composed by the information found and sent to two Velocity templates. Based on the structure defined in Velocity templates and information given by the dictionary, the results are printed to a file in RDF/XML syntax and to an XBRL instance document.

Though the prototype application still can be enhanced, has it succeeded to develop an extraction application where further extensions of the extracted information mainly can be done by adding more words and relations into the ontology. The result is presented in both the RDF and XBRL format, which provides computers with readable information semantically tagged. The result is then considered as knowledge.

One issue that has not been considered in this thesis is when two or more financial figures appear in one sentence. The prototype application is not able to find out which financial words describe which figure. Dealing with semi-structured text this problem is difficult for computers to solve. Several different ways of describing figures are available in natural language; therefore the application needs to analyze the sentence. This can be done by NLP, and helped by ontology specifying word used to indicate which figure it describes.

Abbreviations

4GLs - the Fourth Generation Languages
ANSI - American National Standard Institute
API - Application Programming Interface
CASE - Computer-Aided Software Engineering
COIN - Context Interchange
CWA - Closed World Assumption
DAML - DARPA Agent Markup Language
DOM - Document Object Model
ECOIN - Extended Context Interchange
HP - Hewlett Packard
IE - Information Extraction
IEEE - Institute of Electrical and Electronical Engineers
ISO - International Organization for Standardization
KB - Knowledge Base
NLP - Natural Language processing
NLTK - Natural Language Toolkit
MIT - Massachusetts Institute of Technology
OIL - Ontology Interchange Language
OOPS - Object-Oriented Programming Systems
OWL - Web Ontology Language
PDA - Personal Digital Assistant
RDF - Resource Description Framework
RDFS - Resource Description Framework Schema
RDQL - RDF Query Language
SHOE - Simple HTML Ontology Extensions
SQL - Structured Query Language
URI - Unified Resource Identifier
URL - Unified Resource Locator
XBRL - eXtensible Business Reporting Language
XML - eXensible Markup Language
XSD - XML Schema
XSL - eXtensible Stylesheet Language
VTL - Velocity Template Language

EUR - Euro (€)
GBP - Great Britain pound
NOK - Norwegian krone
USD - United States Dollar ($)

References

[Ala03] Harit Alani, Sanghee Kim, David E. Millard, Mark J. Weal,
Wendy Hall, Paul H. Lewis, and Nigel R. Shadbolt
Automatic Ontology-based Knowledge Extraction from Web documents
University of Southampton,
IEEE January/February 2003, pages 14-21
Also in paper some more detailed by the same authors:
Automatic Ontology-based Knowledge Extraction and Tailored Biography
Generation from the Web,
Intelligence, Agents, Multimedia group, 2003

[Ber96] Tim Berners-Lee
The World Wide Web: Past, Present and Future
http://www.w3.org/People/Berners-Lee/1996/ppf.html

[Ber98] Berners-Lee, T.; Fielding, R.; Irvine, U.C.; Masinter, L.
Uniform Resource Identifiers (URI): Generic Syntax.
IETF Request for Comments: 2396. August 1998
[Online: http://www.ietf.org/rfc/rfc2396.txt]. May 15th

[Ber99] Tim Berners-Lee
Weaving the Web
Harper, San Francisco, USA1999

[Ber01] Tim Berners-Lee, James Hendler, Ora Lassila,
The Semantic Web, W3C.org
Scientific American, May 2001
http://www.w3.org/2001/sw/

[Cri01] Dan Cristea
Anaphora
University of Iasi, EUROLAN – 2001
http://www.racai.ro/EUROLAN-
2001/page/resources/profs/Cristea/Anaphora.ppt, May 15th

[Das02] Subrata Das, Kurt Shuster, Curt Wu
Ontologies for Agent-Based Information Retrieval and Sequence Mining
Cambridge, U.S.A.

[Doa02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy
Learning to Map between Ontologies in the Semantic Web
Proceedings of the World-Wide Web Conference (WWW2002),
ACM Press

[Emb98] Ontology-Based Extraction and Structuring of information from Data-Rich Unstructured Documents
David W. Embley, Douglas M. Campell and Randy D. Smith
Brigham Young Univerity, Provo, U.S.A.

[Fen02] Dieter Fensel
Ontology-Based Knowledge Management
IEEE November 2002, pages 56-59

[Fir02] Aykut Firat, Stuart Madnick and Benjamin Grosof
Knowledge Integration to overcome ontological heterogeneity:
Challenges from financial information systems
Twenty-Third International Conference on Information Systems
http://ebusiness.mit.edu/bgrosof/paps/icis2002-final.pdf, May 22th

[foaf03] FOAF: the 'friend for a friend' vocabulary
http://xmlns.com/foaf/0.1, May 15th

[Gan02] Fabien GANDON
Distributed artificial intelligence and knowledge management:
ontologies and multi-agent systems for a corporate semantic web
nov 2002

[Gar03] Lars Marius Garshol
Living with topic maps and RDF, Ontopia, 2003
http://www.ontopia.net/topicmaps/materials/tmrdf.html#N1961,
April 15. 2003

[Gru93] T.Gruber
A translation Approach to portable ontology specifications
Knowledge acquisition
Vol. 5 1993. 199-220

[Gua98] Guraino N., Welty C.
Formal Ontology and Information Systems
In proceedings of the 1st International Conference on Formal ontologies in Information Systems
FOIS'98, pages 3-15, Italy
IOS Press, June 1998

[Hen03] James Hendler
Science and the Semantic Web
www.sciencemag.org January 23. 2003

[Hil01] Diane Hillmann
Using Dublin Core, 2001
http://dublincore.org/documents/usageguide

[hp03] RDQL – RDF Data Query Language
http://www.hpl.hp.com/semweb/rdql.htm, May 1st

[ITw00] The future of natural-language processing
Unix Insider 12/29/00

[Sow01] John F. Sowa
Building, Sharing, and Merging Ontologies
http://jfsowa.com/ontology/ontoshar.htm

[Klu] Anthony C. Klug, Dennis Tsichritzis:
Multiple View Support within the ANSI/SPARC Framework
pages 477-488 Electronic Edition

[Lau02] Boris Lauser, Tanja Wildemann, Allison Poulos, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz
A Conprehensive Framwork for Building Multilingual Domain Ontologies
DC2002 Florence
http://www.fao.org/agris/aos/Presentations/DC2002.ppt, May 15th

[Mae02] Alexander Maedche and Steffen Stab
ISWC'2002' Tutorial on Ontologies: Representation, Engineering, Learning and Application
University of Karlsruhe 2002

[Ogd23] C. Ogden & I. Richards
The meaning of meaning: A study of the influence, 1923

[Pro101] Natalya F. Noy and Deborah L McGuinness
Ontology Development 101: A Guide to Creating Your first Ontology,
Stanford University, USA,
http://protege.stanford.edu/publications

[Ric02] Jim Richards
An Introduction to XML/XBRL
Murdoch University, Australia 2002
http://www.xbrl.org.au/training/XBRLOverview.pdf

[Ste00] Stefan Decker, Prasenjit Mitra, and Sergey Melnik
Framework of the Semantic Web: An RDF Tutorial
Standford University, 2000,
Published http://computer.org/internet nov 2000

[Tib02] Hendrika Tibbits at University Western Sydney and Jim Richards
Murdoch University
Understanding XBRL
Co- Chairs Education Working Group XBRL Australia Limited
http://www.xbrl.org.au/training/NSWWorkshop.ppt April 23. 2003

[W3Cadr] Naming and Addressing:URIs, URLs, …
http://www.w3.org/Addressing/

[W3Cd+o] DAML+OIL (March 2001) Reference Description
http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218

[W3Cowl] Web Ontology Language (OWL) Use Cases and Requirements
W3C Working Draft 3 February 2003
http://www.w3.org/TR/2003/WD-webont-req-20030331

[W3C99rdf] Resource Description Framework (RDF) Model and Syntax Specification,
W3C Recommendation 22 February 1999
http://www.w3.org/TR/REC-rdf-syntax-19990222

[W3C03rdf] RDF/XML Syntax Specification (Revised),
W3C Working Draft 23 January 2003

[W3Cxml] Extensible Markup Language
http://www.w3.org/XML/

[W3Cvcard] Presenting vCard Objects in RDF/XML
W3C Note 22 February 2001
http://www.w3.org/TR/2001/NOTE-vcard-rdf-20010222

[xbrl] XBRL.org
http://xbrl.org May 14th

[XBRL03] Extensible Business Reporting Language (XBRL) 2.1
Public Working Draft of 2003-04-23
http://www.xbrl.org/2003/XBRL-WD-2003-04-23.pdf
http://www.w3.org/TR/2003/WD-rdf-syntax-grammar-20030123

[XBRL-FR] International Accounting Standards Expressed in XBRL for Electronic Financial Reporting
htt://www.xbrl.org/taxonomy/int/fr/ias, May 16th

Appendix A - Complete RDF for the financial ontology

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
<rdf:RDF xmlns:kb="http://protege.hia.no/ebjoraa/kb#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://protege.stanford.edu/system#"
  xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<kb:AmountUnit rdf:about="http://protege.hia.no/ebjoraa/kb#CurrencyUnit" rdfs:label="CurrencyUnit">
 <kb:synonym xml:space="preserve">
   <![CDATA [&euro; ]]>
  </kb:synonym>
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#ProfitLoss" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#accountAnnualFact" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#company" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeInput" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeMeasure" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeNegative" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeOutput" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financePositive" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financialPapers" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measurePositive" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#payMoney" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#period" />
<kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#predict" />
<kb:synonym>$</kb:synonym>
<kb:synonym>CAD</kb:synonym>
<kb:synonym>DKK</kb:synonym>
<kb:synonym>GBP</kb:synonym>
<kb:synonym>NOK</kb:synonym>
<kb:synonym>SEK</kb:synonym>
<kb:synonym>USD</kb:synonym>
<kb:synonym>USS</kb:synonym>
<kb:synonym>cent</kb:synonym>
<kb:synonym>dollar</kb:synonym>
<kb:synonym>franc</kb:synonym>
<kb:synonym>kr</kb:synonym>
<kb:synonym>krone</kb:synonym>
```

```
  <kb:synonym>ruble</kb:synonym>
    </kb:AmountUnit>
  <a:_instance_annotation rdf:about="http://protege.hia.no/ebjoraa/kb#Finance_00056" a:_creation_timestamp="2003.04.22
    11:48:58.322 CEST" a:_creator="ebjoraa" rdfs:label="Finance_00056" />
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#ProfitLoss" rdfs:label="ProfitLoss">
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeAction" />
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeNegative" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measureNegative" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measurePositive" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#period" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#tax" />
  <kb:synonym>loss</kb:synonym>
  <kb:synonym>profit</kb:synonym>
    </kb:FinanceDescription>
- <kb:ReportFact rdf:about="http://protege.hia.no/ebjoraa/kb#accountAnnualFact" rdfs:label="accountAnnualFact">
  <kb:synonym>annual_account</kb:synonym>
  <kb:synonym>annual_report</kb:synonym>
  <kb:synonym>report</kb:synonym>
  <kb:synonym>result</kb:synonym>
    </kb:ReportFact>
  <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#after_tax" kb:replaceWith="after_tax" kb:synonym="after tax"
    rdfs:label="after_tax" />
  <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#annual_report" kb:replaceWith="annual_report"
    kb:synonym="annual report" rdfs:label="annual_report" />
  <kb:Companies rdf:about="http://protege.hia.no/ebjoraa/kb#area" kb:synonym="cryptograph" rdfs:label="area" />
  <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#before_tax" kb:replaceWith="before_tax" kb:synonym="before
    tax" rdfs:label="before_tax" />
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#billion" kb:replaceWith="B" rdfs:label="billion">
  <kb:synonym>000,000,000</kb:synonym>
  <kb:synonym>000.000.000</kb:synonym>
  <kb:synonym>billion</kb:synonym>
  <kb:synonym>milliard</kb:synonym>
    </kb:TextReplace>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#cent" kb:replaceWith="cent" rdfs:label="cent">
  <kb:synonym>cent</kb:synonym>
  <kb:synonym>cents</kb:synonym>
    </kb:TextReplace>
```

```
- <kb:Companies rdf:about="http://protege.hia.no/ebjoraa/kb#company" kb:synonym="NeoRx" rdfs:label="company">
  <kb:synonym>Amersham</kb:synonym>
  <kb:synonym>Galil</kb:synonym>
    </kb:Companies>
- <kb:StockMarket rdf:about="http://protege.hia.no/ebjoraa/kb#exchangeWords" rdfs:label="exchangeWords">
  <rdfs:comment>Words that occur at the exchange (stock market) when you deal with shares.</rdfs:comment>
  <kb:synonym>leap</kb:synonym>
  <kb:synonym>quote</kb:synonym>
    </kb:StockMarket>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financeAction" rdfs:label="financeAction">
  <kb:synonym>invest</kb:synonym>
  <kb:synonym>operation</kb:synonym>
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financeInput" kb:synonym="purchase"
    rdfs:label="financeInput">
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeOutput" />
  <kb:synonym>bought</kb:synonym>
  <kb:synonym>buy</kb:synonym>
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financeMeasure" rdfs:label="financeMeasure">
  <kb:synonym>cash</kb:synonym>
  <kb:synonym>credit</kb:synonym>
  <kb:synonym>net</kb:synonym>
  <kb:synonym>total</kb:synonym>
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financeNegative" kb:synonym="loss"
    rdfs:label="financeNegative">
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeAction" />
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeNegative" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measureNegative" />
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financeOutput" kb:synonym="sold"
    rdfs:label="financeOutput">
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#financeInput" />
  <kb:synonym>deliver</kb:synonym>
  <kb:synonym>given</kb:synonym>
  <kb:synonym>sale</kb:synonym>
```

```
  <kb:synonym>sell</kb:synonym>
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#financePositive" kb:synonym="revenue"
    rdfs:label="financePositive">
  <kb:synonym>asset</kb:synonym>
  <kb:synonym>income</kb:synonym>
    </kb:FinanceDescription>
- <kb:StockMarket rdf:about="http://protege.hia.no/ebjoraa/kb#financialPapers" rdfs:label="financialPapers">
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#exchangeWords" />
  <kb:synonym>first call</kb:synonym>
  <kb:synonym>fund</kb:synonym>
  <kb:synonym>share</kb:synonym>
  <kb:synonym>stock</kb:synonym>
    </kb:StockMarket>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#first_quarter" kb:replaceWith="first_quarter"
    rdfs:label="first_quarter">
  <kb:synonym>1st quarter</kb:synonym>
  <kb:synonym>first quarter</kb:synonym>
    </kb:TextReplace>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#fourth_quarter" kb:replaceWith="fourth_quarter"
    rdfs:label="fourth_quarter">
  <kb:synonym>4th quarther</kb:synonym>
  <kb:synonym>fourth quarter</kb:synonym>
    </kb:TextReplace>
- <kb:Transaction rdf:about="http://protege.hia.no/ebjoraa/kb#getMoney" rdfs:label="getMoney">
  <kb:synonym>get</kb:synonym>
  <kb:synonym>receive</kb:synonym>
    </kb:Transaction>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#measureNegative" kb:synonym="reduction"
    rdfs:label="measureNegative">
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measurePositive" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#percentChange" />
  <kb:synonym>decline</kb:synonym>
  <kb:synonym>decrease</kb:synonym>
  <kb:synonym>down</kb:synonym>
  <kb:synonym>fall</kb:synonym>
  <kb:synonym>fell</kb:synonym>
```

```
  <kb:synonym>impair</kb:synonym>
  <kb:synonym>loose</kb:synonym>
  <kb:synonym>losing</kb:synonym>
  <kb:synonym>lost</kb:synonym>
  <kb:synonym>recess</kb:synonym>
    </kb:FinanceDescription>
- <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#measurePositive" rdfs:label="measurePositive">
  <kb:inverseInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#measureNegative" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#percentChange" />
  <kb:synonym>benefit</kb:synonym>
  <kb:synonym>gain</kb:synonym>
  <kb:synonym>grow</kb:synonym>
  <kb:synonym>increase</kb:synonym>
    </kb:FinanceDescription>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#million" kb:replaceWith="M" rdfs:label="million">
  <kb:synonym>,000,000</kb:synonym>
  <kb:synonym>000,000</kb:synonym>
  <kb:synonym>000.000</kb:synonym>
  <kb:synonym>million</kb:synonym>
    </kb:TextReplace>
- <kb:TimeUnit rdf:about="http://protege.hia.no/ebjoraa/kb#month" rdfs:label="month">
  <kb:synonym>april</kb:synonym>
  <kb:synonym>august</kb:synonym>
  <kb:synonym>december</kb:synonym>
  <kb:synonym>february</kb:synonym>
  <kb:synonym>january</kb:synonym>
  <kb:synonym>july</kb:synonym>
  <kb:synonym>june</kb:synonym>
  <kb:synonym>march</kb:synonym>
  <kb:synonym>may</kb:synonym>
  <kb:synonym>november</kb:synonym>
  <kb:synonym>october</kb:synonym>
  <kb:synonym>september</kb:synonym>
    </kb:TimeUnit>
- <kb:Transaction rdf:about="http://protege.hia.no/ebjoraa/kb#payMoney" kb:synonym="payment" rdfs:label="payMoney">
  <kb:synonym>expense</kb:synonym>
  <kb:synonym>pay</kb:synonym>
```

```
    </kb:Transaction>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#percent" kb:replaceWith="%" kb:synonym="prosent"
    rdfs:label="percent">
  <kb:synonym>%</kb:synonym>
  <kb:synonym>percent</kb:synonym>
    </kb:TextReplace>
  <kb:AmountUnit rdf:about="http://protege.hia.no/ebjoraa/kb#percentChange" kb:synonym="%" rdfs:label="percentChange" />
- <kb:TimeUnit rdf:about="http://protege.hia.no/ebjoraa/kb#period" kb:synonym="third_quarter" rdfs:label="period">
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#month" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#period_info" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#year" />
  <kb:synonym />
  <kb:synonym>annual</kb:synonym>
  <kb:synonym>first_quarter</kb:synonym>
  <kb:synonym>fourth_quarter</kb:synonym>
  <kb:synonym>period</kb:synonym>
  <kb:synonym>second_quarter</kb:synonym>
    </kb:TimeUnit>
- <kb:TimeUnit rdf:about="http://protege.hia.no/ebjoraa/kb#period_info" kb:synonym="end" rdfs:label="period_info">
  <kb:synonym>after</kb:synonym>
  <kb:synonym>before</kb:synonym>
    </kb:TimeUnit>
- <kb:Prediction rdf:about="http://protege.hia.no/ebjoraa/kb#predict" kb:synonym="prognos" rdfs:label="predict">
  <kb:synonym>anticipat</kb:synonym>
  <kb:synonym>approximat</kb:synonym>
  <kb:synonym>belief</kb:synonym>
  <kb:synonym>claim</kb:synonym>
  <kb:synonym>estimate</kb:synonym>
  <kb:synonym>expect</kb:synonym>
  <kb:synonym>forecast</kb:synonym>
  <kb:synonym>forward-looking</kb:synonym>
  <kb:synonym>intent</kb:synonym>
  <kb:synonym>predict</kb:synonym>
    </kb:Prediction>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#second_quarter" kb:replaceWith="second_quarter"
    kb:synonym="second quarter" rdfs:label="second_quarter">
  <kb:synonym>2nd quarter</kb:synonym>
```

```
  <kb:synonym>andre kvartal</kb:synonym>
    </kb:TextReplace>
  <kb:FinanceDescription rdf:about="http://protege.hia.no/ebjoraa/kb#tax" kb:synonym="tax" rdfs:label="tax" />
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#textReplace" rdfs:label="textReplace">
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#after_tax" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#annual_report" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#before_tax" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#billion" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#cent" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#first_quarter" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#fourth_quarter" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#million" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#percent" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#second_quarter" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#third_quarter" />
  <kb:refInstance rdf:resource="http://protege.hia.no/ebjoraa/kb#thousand" />
    </kb:TextReplace>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#third_quarter" kb:replaceWith="third_quarter"
    rdfs:label="third_quarter">
  <kb:synonym>3rd quarter</kb:synonym>
  <kb:synonym>third quarter</kb:synonym>
    </kb:TextReplace>
- <kb:TextReplace rdf:about="http://protege.hia.no/ebjoraa/kb#thousand" kb:replaceWith="T" rdfs:label="thousand">
  <kb:synonym>,000</kb:synonym>
  <kb:synonym>000</kb:synonym>
  <kb:synonym>thousand</kb:synonym>
  <kb:synonym>tusen</kb:synonym>
    </kb:TextReplace>
- <kb:TimeUnit rdf:about="http://protege.hia.no/ebjoraa/kb#year" kb:synonym="2004" rdfs:label="year">
  <kb:synonym>2000</kb:synonym>
  <kb:synonym>2001</kb:synonym>
  <kb:synonym>2002</kb:synonym>
  <kb:synonym>2003</kb:synonym>
    </kb:TimeUnit>
    </rdf:RDF>
```

Appendix B - Complete RDFS for the financial ontology

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
- <rdf:RDF
  xmlns:kb="http://protege.hia.no/ebjoraa/kb#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://protege.stanford.edu/system#"
  xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#AmountUnit" rdfs:label="AmountUnit">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Area" rdfs:label="Area">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Companies" rdfs:label="Companies">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  </rdfs:Class>
  <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#CurencyUnit_1" rdfs:label="CurencyUnit_1" />
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Facts" rdfs:label="Facts">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Finance" rdfs:label="Finance">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#FinanceDescription" rdfs:label="FinanceDescription">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" rdfs:label="MeasurementUnit">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Prediction" rdfs:label="Prediction">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Report" rdfs:label="Report">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Area" />
  </rdfs:Class>
```

```
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#ReportFact" rdfs:label="ReportFact">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Facts" />
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#StockMarket" />
    </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#StockMarket" rdfs:label="StockMarket">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Area" />
    </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#TextReplace" rdfs:label="TextReplace">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#TimeUnit" rdfs:label="TimeUnit">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
    </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#Transaction" rdfs:label="Transaction">
  <rdfs:subClassOf rdf:resource="http://protege.hia.no/ebjoraa/kb#Area" />
    </rdfs:Class>
- <rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/kb#WordNet" rdfs:label="WordNet">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Class" />
    </rdfs:Class>
- <kb:WordNet rdf:about="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" rdfs:label="WordNet_ROOT">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </kb:WordNet>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#child-of" rdfs:label="child-of">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#definition" rdfs:label="definition">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#inverseInstance" rdfs:label="inverseInstance">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#morphological_form" rdfs:label="morphological_form">
```

```
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#name" a:maxCardinality="1" rdfs:label="name">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#pattern" rdfs:label="pattern">
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#ref1" rdfs:label="ref1">
  <rdfs:subPropertyOf rdf:resource="http://protege.hia.no/ebjoraa/kb#CurencyUnit_1" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#refClass" a:range="cls" rdfs:label="refClass">
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Class" />
  <a:allowedParents rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#refInstance" rdfs:label="refInstance">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#AmountUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#TextReplace" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#replaceText" rdfs:label="replaceText">
  <a:allowedClasses rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#TextReplace" />
  <a:allowedClasses rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#replaceWith" a:maxCardinality="1" rdfs:label="replaceWith">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#TextReplace" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
```

```
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#synonym" rdfs:label="synonym">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#TextReplace" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#WordNet_ROOT" />
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.hia.no/ebjoraa/kb#type" rdfs:label="type">
  <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
    </rdf:Property>
- <rdf:Property rdf:about="http://protege.stanford.edu/system#_name" rdfs:label=":NAME">
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#Finance" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#MeasurementUnit" />
  <rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/kb#TextReplace" />
    </rdf:Property>
    </rdf:RDF>
```

Appendix C - Complete RDFS for the article ontology

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:article="http://protege.hia.no/ebjoraa/article#"
  xmlns:a="http://protege.stanford.edu/system#" xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
<rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/article#Article" rdfs:label="Article">
<rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  </rdfs:Class>
<rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/article#Financial_tems" rdfs:label="Financial_tems">
<rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  </rdfs:Class>
<rdfs:Class rdf:about="http://protege.hia.no/ebjoraa/article#Sentences" rdfs:label="Sentences">
<rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Resource" />
  </rdfs:Class>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#URL" a:maxCardinality="1" rdfs:label="URL">
<rdfs:comment>ID of the article. Uniqe referance to an articles publication area.</rdfs:comment>
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#currencyAmount" a:maxCardinality="1"
  rdfs:label="currencyAmount">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Financial_tems" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#currencyUnit" a:maxCardinality="1"
  rdfs:label="currencyUnit">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Financial_tems" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#financial_ItemID" a:maxCardinality="1" a:range="integer"
  rdfs:label="financial_ItemID">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Financial_tems" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#language" a:maxCardinality="1" rdfs:label="language">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
```

```
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#part-of" a:maxCardinality="1" a:minCardinality="1"
  a:range="cls" rdfs:label="part-of">
<a:allowedParents rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Financial_tems" />
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Sentences" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Class" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#pubdate" a:maxCardinality="1" rdfs:label="pubdate">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#sentenceID" a:maxCardinality="1" a:minCardinality="1"
  a:range="integer" rdfs:label="sentenceID">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Sentences" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#source" a:maxCardinality="1" rdfs:label="source">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#subject" a:maxCardinality="1" rdfs:label="subject">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#text" a:maxCardinality="1" rdfs:label="text">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Sentences" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#title" a:maxCardinality="1" rdfs:label="title">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#wordType" rdfs:label="wordType">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Sentences" />
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
```

```
<rdf:Property rdf:about="http://protege.hia.no/ebjoraa/article#words" rdfs:label="words">
<rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal" />
  </rdf:Property>
<rdf:Property rdf:about="http://protege.stanford.edu/system#_name" rdfs:label=":NAME">
<rdfs:domain rdf:resource="http://protege.hia.no/ebjoraa/article#Article" />
  </rdf:Property>
  </rdf:RDF>
```

Appendix D - Java code for parsing XML input documents for information

```
package FinanceExtr;
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class getTextFromXML
      {
      public static String getString(String myFile, String myNode) throws Exception
            {
              //The file to read and which node to find is given as input parameter
              String myStr = "null";  //"null" is returned of the tag is not found or containing no text
              DocumentBuilderFactory factory=javax.xml.parsers.DocumentBuilderFactory.newInstance();
              DocumentBuilder builder=factory.newDocumentBuilder();
              org.w3c.dom.Document doc=builder.parse(new FileInputStream(myFile));

              NodeList list=doc.getElementsByTagName(myNode);  //Go through every node, finds right node by tag name
              for(int i=0;i<list.getLength();++i)
                  {
                     myStr = FindNodes(list.item(i),0);
                  }
              return myStr; //Return text found for given tag to Jython
            }//end getString

      static String FindNodes(Node node,int level)
            {
              String nodeText = "";
              if(node.getNodeValue()!=null)
                  {
                     nodeText = node.getNodeValue().trim();
                     //Gets the value of the node when found
                  }
              else
                  {  System.out.print(); }

              NodeList childnodes=node.getChildNodes();
              for(int i=0;i<childnodes.getLength();++i)
                  {
                    nodeText  = FindNodes(childnodes.item(i),level+1);
                  }
              return nodeText; //Return text found for the tag to getString
            }//end findNodes
}//end class
```

Appendix E -

Java code for parsing and querying the financial ontology

```
package FinanceExtr;

import com.hp.hpl.mesa.rdf.jena.model.* ;
import com.hp.hpl.mesa.rdf.jena.mem.* ;
import com.hp.hpl.mesa.rdf.jena.common.* ;
import com.hp.hpl.jena.rdf.query.* ;
import java.util.* ;
import java.io.* ;

public class getInputFromOnt
{
    //reads the ontology, finds different synonyms for an instance
    // and returns a vector of all the synonyms that were found
    // param1: myFile     - which ontology file to read
    // param2: myInstance - URI for current instance to search for
    // param3: myResURL   - URI for the ontology
    // param3: myResURL   - which slot (i.e. synonym or refInstance) to find in the instance
      public Vector getSynonyms(String myFile,String myURL,String myResURL,String mySlot)
      {

       String myString  = "test";
       String myString2 = "test";
       String retString = "test";
       Vector v = new Vector();

       try {

          Model model = new ModelMem() ;
          model.read(new FileReader(myFile),
                   "http://nowhere/",
                   "RDF/XML") ;

          //Declaration of the query
          String queryString = "SELECT ?x, ?resource,?z "+
          "WHERE (<"+myURL+">,<Finance:"+mySlot+">, ?z) "+
          "USING Finance FOR <"+myResURL+">";

          Query query = new Query(queryString) ;
          query.setSource(model);
          //the query is executed
```

```
          QueryExecution qe = new QueryEngine(query);

            //execute query
           QueryResults results = qe.exec();

           for ( Iterator iter = results ; iter.hasNext(); )
           {
                  ResultBinding res = (ResultBinding)iter.next() ;
                  Object z = res.get("z") ;

                  myString = z.toString();

                  boolean isFound = v.contains(myString);
                  if (isFound == true)
                   {
                       //break;
                   }//end if
                  else
                   {
                       v.addElement(myString);
                   }//end else

             }//end for

             results.close();

           }//end try
          catch (Exception ex)
           {
             System.err.println("Exception: "+ex) ;
             ex.printStackTrace(System.err) ;
           }//end catch

         return v;

      }//end getSynonyms
}//end class
```

Appendix F -

Prototype application agent code - Jython code for extracting financial information

```
from re import *
from nltk.mytoken import *
from nltk.tagger import *
#Java files query for text from input file and ontology
from FinanceExtr import getInputFromOnt, getTextFromXML
from java.util import *

import string,os, sys
import org.apache.velocity.app.Velocity as Velocity
import org.apache.velocity.Template as Template
import org.apache.velocity.VelocityContext as VelocityContext
import java.io.StringWriter as StringWriter
import java.util.ArrayList as ArrayList
import java.util.Properties as Properties

myVelocity_RDF  = 'RDF.vm'          #velocity tamplate file used to print output in RDF
myVelocity_XBRL = 'XBRL.vm'         #velocity tamplate file used to print output in XBRL
myRDF_file      = 'Finance.rdf'     #Finance ontology file
myRDF_KB        = 'http://protege.hia.no/ebjoraa/kb#'                #URI for ontology, used in query
RDFcurrencyUnit = 'http://protege.hia.no/ebjoraa/kb#CurrencyUnit'    #start point for ontology query
myRDFSchema     = 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#' #RDF Schema for ontology

class Template:
      def __init__(self, templatename):

            p = Properties()
            p.setProperty("runtime.log.logsystem.class", "org.apache.velocity.runtime.log.NullLogSystem")
            Velocity.init(p)

            self.t = Velocity.getTemplate(templatename)

      def py2java(self,v):
            if type(v)==type(""):
                  return v
            elif type(v) in [type([]),type(())]:
                  l = ArrayList()
                  for e in v: l.add(self.py2java(e))
                  return l
            elif type(v)==type({}):
                  m = Hashtable()
```

```
                for e,i in v.items(): m.put(e,self.py2java(i))
                return m
          else:
                #print 'Type is not supported:',type(v)
                return None
    def fill(self, data):
          context = VelocityContext()
          for k,v in data.items():
                context.put(k,self.py2java(v))
          w = StringWriter()
          self.t.merge(context, w)
          return w

class InstanceList:
    def _init_(self):
          l          = [] #all resources checked
          refs       = [] #all resources to be checked
          sentences  = [] #all sentences, information
          currencies = [] #all currencies in one sentence
          myDict     = {} #dict to velocity template
          myDict2    = {} #dict to velocity template
          pass
    def setList(self,myIn,c):
          c.append(myIn)
          return c
    def getList(self,c):
          return c
    def getListItem(self,i,c):
          return c[i]
    def getLength(self,c):
          return len(c)
    def delList1(self,c,i):
          if i == 1:
                try:
                      theListItem = c[0]
                      c.remove(theListItem)
                except:
                      pass
          else:
                i = i-1
                try:
                      theListItem = c[i]
                      c.remove(theListItem)
                except:
                      pass
```

```
      def delList(self,c):
            lengthList = len(c)
            for i in range(lengthList):
                  theListItem = c[0]
                  c.remove(theListItem)
      def updateDict(self,myDictIn):
            myDict.update(myDictIn)

class TextInOut:
      def textInOutFromInputXML(self,myFile,myNode):
            #myFile - which RDF URI/node to search for, usually like myRDF_KB
            #myNode - information to get
            textInOut  = getTextFromXML() #making an object of type getTextFromXML from Java
            textString = textInOut.getString(myFile,myNode) #calls getString function in getTextFromOnt.class
            return textString #returns a string

      def getWordsFromOnt(self,myNS,instance,mySlot):
            #input er URL til resource og type slot som en vil finne (synonym|reference)
            #myNS     - URI namespace to use
            #instance - which instance to query for in the ontology, like "CurrencyUnit"
            #mySlot   - which slot (property) of queried instance you want to find, like synonym or refInstance
            getInputSyn = getInputFromOnt() #making an object of type getInputFromOnt from Java
            #calls getSynonyms function in getInputFromOnt.class
            synVector = getInputSyn.getSynonyms(myRDF_file,instance,myNS,mySlot)
            return synVector #returns a vector of the result

      def printToFile(self,text,ending):
            #prints to file specified in cmd line
            if len(sys.argv) > 2:
                  output=open(sys.argv[2]+ending,'a')
                  output.writelines(text)
                  output.close()
class TextChange:
      def textReplace(self,text):
            #text to do modification on is given as input
            textChange = TextChange()         #new instance of class
            text = text.replace('\"','\'')    #replace all " with ' because of XML syntax
            text = text.replace('.\n','. \n') #replace all '.' followed by a line enters with a white space
            text = text.replace('\n','')      #removes all \n from the text, in python this is a new line
            text = text.replace('--','')      #replace all '--' with '-' because this is comment in XML


            r=re.compile('[A-Z]\.[A-Z]\.([A-Z]{2,})')
            text = r.sub(';\\1',text)         #replace all U.S. with U.S; to avoid misinterpreted sentence endings
```

```
        r=re.compile('\.([0-9]{1,})')
        text = r.sub(',\\1',text)
        #all numbers of format XX.XX are replaced with XX;XX

        r=re.compile('\.([^A-Z|\'|!]{2,})')
        text = r.sub(';\\1',text)
        #replace all '.' with ';' which are not followed by a capital letter or ' in the 2 first signs

        text = textChange.replaceTextFromOnt(text) #calls def replaceTextFromOnt
        return text

    def replaceTextFromOnt(self,text):
        #Words given in the ontology defined to be replaced are found and changed here
        textInOut = TextInOut() #new instance of class
        myRef     = textInOut.getWordsFromOnt(myRDF_KB,myRDF_KB+'textReplace','refInstance')
        #gets URL to all words to be replaced from the ontology

        nbRef     = len(myRef) #number of referances to words to be changed
        for i in range(nbRef):
            #loops through every instance, finding word to replace, and what to replace with
            mySyn       = textInOut.getWordsFromOnt(myRDF_KB,myRef[i],'synonym')
            #vector of words(synonyms) to replace
            textReplace = textInOut.getWordsFromOnt(myRDF_KB,myRef[i],'replaceWith')
            #replace with this word
            nbSyn       = len(mySyn)
            for j in range(nbSyn):
                if textReplace[0] == 'M' or textReplace[0] == 'B' or textReplace[0] == 'cent':

                    text  = text.replace(' '+mySyn[j],textReplace[0])
                    #takes those with space between them and the word to replace, ex  $50 million -> $50M
                    text  = text.replace(mySyn[j],textReplace[0])
                    #takes those with space between them and the word to replace, ex  $50 million -> $50M
                else:
                    text  = text.replace(mySyn[j],textReplace[0])
                    #takes those with space between them and the word to replace, ex  $50,000,000 -> $50M
        return text #return the text with changes

    def myTagger(self,textstr):
        """
        Tagger function taken from NTLK toolkit, tagger.py demo().
        Some changes has been made.
        This calls the tagger of myNN_CD_Tagger. A modified CD(number) and NN (word)
        tagger from NLTK.
        """
```

```
            tokens = TaggedTokenizer().tokenize(textstr)
            train_tokens = tokens[:]
            test_tokens  = tokens[:]

            #print 'training unigram tagger (%d tokens)...' % len(train_tokens)
            t0 = UnigramTagger()
            t0.train(tokens)

            #print 'training 1st order tagger (%d tokens)...' % len(train_tokens)
            t1 = NthOrderTagger(1)
            t1.train(tokens)

            #print 'training 2nd order tagger (%d tokens)...' % len(train_tokens)
            t2 = NthOrderTagger(2)
            t2.train(tokens)

            #print 'creating combined backoff tagger...'
            ft    = BackoffTagger( (t2, t1, t0, myNN_CD_Tagger()) )
            strFT = str(ft)

            #print 'running the tagger... (%d tokens)...' % len(tokens)
            result = ft.tag(untag(tokens))
            #print 'Accuracy: %.5f' % accuracy(tokens, result)
            return result

class TextExtract:
      def findNumbInSentence(self,text):
            #this function is called only the first time in every sentence, for finding currencies
            instanceList  = InstanceList() #new instance of class
            textInOut     = TextInOut()    #new instance of class
            textChange    = TextChange()   #new instance of class
            textExtract   = TextExtract()  #new instance of class

            tokens = MyLineTokenizer().mytokenize(text) #Tokenize text into sentence tokens
            lenTok = len(tokens) #number of tokens

            mySyn = textInOut.getWordsFromOnt(myRDF_KB,RDFcurrencyUnit,'synonym')
            #get vector of all currencies in the ontology

            nbSyn = len(mySyn) #number of synonyms
            Found = '0' #a variable preventing the loop to enter findWithRefSyn without having found an currencyunit
            myInt = 0   #sentence number

            for token in tokens: #for each sentence in article
                  myDict       = {}  #declaration of dictionary for words found in sentence
```

```
myDict2      = {}
myDictcuca   = {}  #declaration of dictionary for currencies and amount found in sentence
myList       = []  #new list
FoundAt      = []  #list variabel to hold where info was found, not implemented
print '------------------NEXT SENTENCE----------------------------'
thisSentence = token.type() #variable containing the text of the current sentence
instanceList.delList(l)     #deleting the old list of instances checked (loop preventer)

myInt = myInt + 1
myDict['SID']   = str(myInt)   #Sentence ID   is added to dict
myDict['SText'] = thisSentence #Sentence Text is added to dict

tagResult = textChange.myTagger(thisSentence)
#function for tagging sentence, ex format: 'ended'/'NN'@[1w] and '2002'/'CD'@[4w]

tagResLen = len(tagResult) #number of words in tagged sentence

myWordToken =  WSTokenizer().tokenize(thisSentence)
#tokenizing the same sentence without CD and NN tags

myCDInt = 0
for myStr2 in range(tagResLen): # for each word in sentence
      thisWord = (myWordToken[myStr2].type())    #the current word cheched without CD/NN tag
      thisWordLoc =  (myWordToken[myStr2].loc())
      #the current words location in text (without CD/NN tag), not implemented

      tagWord = str(tagResult[myStr2].type())    #the current word cheched with CD/NN tag
      a = 1

      y={}
      try: #Find numbers(CD) in sentence/token
            expr = re.compile('CD')
            p    = re.findall(expr, tagWord)
            if p == ['CD']:  #a number(CD tag) is found
                  sentenceToken = 's'
                  if not sentenceToken in tagWord:
                        #check all words without the one '::' denoting end of sentence
                        #myDictcuca.clear()

                        #Knows its a number, has is a belonging currencyUnit
                        for i in range(nbSyn): #go through the vector of currencies found above
                              myV=str(mySyn[i])       #myV is current currency
                              e=0                  #e denote euro, if e=1 euro(€) is found
                              #print repr(myV), thisWord
```

```
if myV == '&euro;':
      #Euro from ontology (or '&#8364;':)
      myV = unichr(0x20ac)
      #convert to unichar of euro, because of the text has been changed
try:  #check if thisWord is Euro
      pxml = re.compile(myV)
      pont = pxml.search(thisWord)
      if pont:
            if repr(myV) == repr(unichr(0x20ac)):
                  e=1         #Euro found
except:
      pass                    #Euro not found

if myV == '$':
      expr = re.compile('\\$')
      #$ is a special symbol regular expr and '\\'
      #is used to for stating that $ = dollar
      p    = re.findall(expr, tagWord)
else:
      expr = re.compile(myV)
      #find match between currencies in onto an thisWord from the article
      p    = re.findall(expr, tagWord)
if p == [myV] or e == 1:
      myCDInt = myCDInt + 1
      #keep track of currencies when more than one occur in one sentence

      #currencyUnit is found, go to ontologo for refInstances
      Found = '1'
      #ok to go to the findWithRefSyn-loop and find words belonging to
      # the financial number found
      thisWord = str(thisWord)
      if e == 1:
            print 'ca', str(thisWord)
            thisWordtemp = re.sub(unichr(0x20ac),'', thisWord)
            #removes € from thisWord, finds amount

            thisWordtemp = re.sub('''[^0-9A-Z]''''',','', thisWord)
            #thisWordtemp = re.sub('.','', thisWord)
            myDictcuca['CU'] = 'eur'
            #currencyUnit €=EUR for ISO4217 uniformity

            #myDict['CU'] = '&#8364;'
            #currencyUnit (Euro)   is added to dict
            print 'ca', str(thisWordtemp),'ca'
            myDictcuca['CA'] = str(thisWordtemp)
```

```
                                        #currencyAmount ex 56M is added to dict
                                  else:
                                        if myV == '$':
                                              myV = 'usd' #for ISO4217 uniformity
                                              #thisWordtemp = re.sub('\\$','', thisWord)
                                              #need to be done for python to use the sign $ and not
                                              #internal use
                                              thisWordtemp = re.sub('\\$','', thisWord)
                                              myDictcuca['CU'] = myV
                                              #currencyUnit, ex GBP  is added to dict
                                              myDictcuca['CA'] = str(thisWordtemp)
                                              #currencyAmount ex 56M is added to dict

                                        else:
                                              thisWordtemp = re.sub('.','', thisWord)
                                              thisWordtemp = re.sub(myV,'', thisWord)


                                              myDictcuca['CU'] = myV
                                              #currencyUnit, ex GBP  is added to dict
                                              myDictcuca['CA'] = str(thisWordtemp)
                                              #currencyAmount ex 56M is added to dict
                                  myList.append(myDictcuca)
                                  myDictcuca={}
                                  break #match found, no need to check others

                  else:
                        expr = re.compile('NN')
                        p    = re.findall(expr, tagType)
            except:
                  i=1#pass
                  #The word was not a number/CD
      if Found == '1': #currency is found and call findRefWithSyn to get more information
            print 'Found'
            myDict['CuCa']= myList
            instanceList.updateDict(myDict)
            print 'myDict ',myDict
            textExtract.findWithRefSyn(tagResult,tagResLen,thisWord,myDict,myInt)
            #calls findWithRefSyn, necessary parameters are added
            Found = '0' #prevents the loop to go inside findWithRefSyn without finding of currency
      else:
            print 'Not found'
            #add the sentence where no financal info was found to list
            instanceList.delList1(sentences,myInt)
            #avoids duplicates, delete a former version of this dict in list
```

```
                instanceList.setList(myDict,sentences) #add new dict to list
        return lenTok


    def findWithRefSyn(self,tagResult,tagResLen,thisWord,myDict,myInt):
        #go to reference of instance and find its synonyms,
        #checks against every word in every sentence currency is found
        print '------- Currency Found in sentence -------'
        instanceList  = InstanceList() #new instance of class
        instanceList  = []             #new list
        firstLevelRef = []             #new list, first level references
        nextLevelRef  = []             #new list, references from instances in firstLevelRef
        textExtract   = TextExtract()  #new instance of class

        myRef = textExtract.findRefInstances(RDFcurrencyUnit,1)
        #find all references to other instances from the current instance
        nbRef = len(myRef) #number of references

        for j in range(nbRef): #find synonyms and refInstances to all the instances related to currencyUnit
            firstLevelRef = textExtract.findSyn(myRef[j],tagResult,tagResLen,myDict,myInt)
            #checks in findSyn for synonyms
            try:
                listLen = len(firstLevelRef)
                if listLen > 0:
                    w = -1  #temp var
                    while w < len(firstLevelRef):
                        #while there are more instances to check for synonyms and refInstance
                        w=w+1
                        nextLevelRef = textExtract.findSyn(firstLevelRef[w],tagResult,tagResLen,myDict,myInt)
                        #checks in findSyn for synonyms
                        if len(nextLevelRef) > 0: #if a list of refInstances is returned, add to list
                            for xx in range(len(nextLevelRef)): #add all returned
                                firstLevelRef.append(nextLevelRef[xx])
                                #the list is updated if synonym is found and refInstances given
            except:
                pass

    def findSyn(self,myRef,tagResult,tagResLen,myDictIn,myInt):
        instanceList = InstanceList() #new instance of class
        textInOut    = TextInOut()    #new instance of class
        textChange   = TextChange()   #new instance of class
        textExtract  = TextExtract()  #new instance of class
        RFoundAt     = []             #new list, not inplemented, to find relations
                                      #between where words are in sentence and compared to other words
        RFound       = 0
```

```
        myDict       = myDictIn
        newRefI      = []

        thisInstSynonyms  = textInOut.getWordsFromOnt(myRDF_KB,myRef,'synonym')
        #gets a vector of synonyms for current instance from ontology
        thisInstName      = textInOut.getWordsFromOnt(myRDFSchema,myRef,'label') #gets name of instance
        nbSynRef          = len(thisInstSynonyms)     #number of synonyms

        for m in range(nbSynRef): #for all synonyms
              thisSynonym = str(thisInstSynonyms[m]) #current synonym
              for n in range(tagResLen): #check every words in sentence for match of current synonym
                    theWord =  myUntagger(tagResult,n) #untagger strips the word for CD/NN

                    expr = re.compile(thisSynonym,re.IGNORECASE)
                    p    = re.findall(expr, theWord)   #using reg expr for finding match

                    if p == [thisSynonym]: #if match is found
                          RFound  = 1       #var denoting match found

                          if RFoundAt == []: RFoundAt = [n]  #list telling where match found, not implemented further
                          else: RFoundAt = RFoundAt+[n]
                          theWord = theWord.replace('.','')
                          #replaces '.' with a white space from the found word if this exist
                          if len(thisInstName[0]) > 2:
                                thisInstName         = thisInstName[0]
                          #thisInstName is taken from a vector, to avoid '[' and ']' the first instance is collected
                          myDict[thisInstName] = theWord
                          newRefI = textExtract.findRefInstances(myRef,1)
                          #does this instance have a reference to other instances?

        instanceList.delList1(sentences,myInt) #avoids duplicates, delete a former version of this dict in list
        instanceList.setList(myDict,sentences) #add new dict to list
        if RFound == 1:
              return newRefI   #no ref returned to FindWithRefSyn
        else:
              return  newRefI  #list of refs is returned to FindWithRefSyn

  def findRefInstances(self,RDF_resource,param):
        textInOut    = TextInOut()   #new instance of class
        textExtract  = TextExtract() #new instance of class
        tempList     = []            #new list
        refInstances = textInOut.getWordsFromOnt(myRDF_KB,RDF_resource,'refInstance')
        #gets all refInstances for this instance
        nbRef = len(refInstances) #number of references to other instances
        for i in range(nbRef):
```

```
                ok = textExtract.checkList(refInstances[i])
                #checks for every instance in checklist is synonyms already is found (loop preventer)
                if ok == 1 and param == 0:
                      return refInstances[i]
                elif ok == 0 and param == 0:
                      return 0
                elif ok == 1 and param == 1:
                      tempList.append(refInstances[i])
          return tempList

    def checkList(self,thisInstance):
          instanceList = InstanceList() #new instance of class
          a = 1  #temp var for found in list of instances found synonyms for: 1=not found 0=found
          theList    = instanceList.getList(l)
          theListLen = len(theList)
          for i in range(theListLen):        #loops through list and checks if already found synonyms
                if thisInstance == theList[i]: #checks every instance is in list
                      a = 0 #instance is in list
                      break
                else:
                      a = 1 #instance is not in list
          if a == 1:    #If a==1, instance not in list -> add to list for finding synonyms
                lSet = instanceList.setList(thisInstance,l)
          return a      #returns 1 or 0 to findRefInstances

if __name__ == '__main__':
    #import sys
    try:
          templateVelocityRDF  = Template(myVelocity_RDF)
          templateVelocityXBRL = Template(myVelocity_XBRL)
    except:
          print '--- Velocity file not found. ---'
          raise SystemExit
    myDict  = {}
    instanceList = InstanceList()
    l          = [] #alle resources som er sjekket
    refs       = [] #alle resources som skal sjekkes
    sentences  = [] #all sentences, information
    currencies = [] #all currencies in one sentence

    textInOut   = TextInOut()
    textChange  = TextChange()
    textExtract = TextExtract()
    try:
          fileName = sys.argv[1]
```

```
except:
      print '''File name not given, like 'python xx.py text.txt' '''
      raise SystemExit

textDate = textInOut.textInOutFromInputXML(fileName,'pubDate')
textSource = textInOut.textInOutFromInputXML(fileName,'Source')
textSubject = textInOut.textInOutFromInputXML(fileName,'Subject')
textLang = textInOut.textInOutFromInputXML(fileName,'language')
textURL = textInOut.textInOutFromInputXML(fileName,'URL')
textStrings = textInOut.textInOutFromInputXML(fileName,'Text')

tYear  = textDate[0:4]       #find this year article published
tPrevY = str(int(tYear) - 1) #find last year
tMonth = textDate[4:6]       #find month article published
tDay   = textDate[6:8]       #find day   article published

myDict = {'fileName':fileName,'myRDF_file':myRDF_file, 'URL':textURL, 'Language':textLang,
             'pubDate':textDate,'Subject':textSubject,'Source':textSource,
             'Year':tYear,'Pyear':tPrevY,'Month':tMonth,'Day':tDay}
instanceList.updateDict(myDict)

textString = textChange.textReplace(textStrings)

mySentenceNR = textExtract.findNumbInSentence(textString)
sentences=instanceList.getList(sentences)
myDict['sentences'] = sentences
print 'myDict->', myDict

#-- RDF output
templateTextRDF = templateVelocityRDF.fill(myDict)
#print templateTextRDF
textInOut.printToFile(str(templateTextRDF),'.rdf')

#-- XBRL output
templateTextXBRL = templateVelocityXBRL.fill(myDict)
templateTextXBRL = str(templateTextXBRL)
textInOut.printToFile(templateTextXBRL,'.xml')
print '''---------------- End this run -----------------------'''
```

Appendix G –

Changes made in NLTK toolkit in the files token.py and tagger.py

Token.py

```
class MyLineTokenizer(TokenizerI):
    """
    A tokenizer that separates a string of text into sentences, based
    on newline characters.  Each sentence is encoded as a C{Token}
    whose type is a C{string}.  Location indices start at zero, and
    have a unit of C{'s'}.
    """
    def __init__(self): pass

    def mytokenize(self, str, source=None):
        assert _chktype(1, str, types.StringType)
        tokens = []
        i = 0

        for sent in str.split('. ') or str.split('.\n') or str.split('.\' ') or str.split('\n\n') or str.split('?'):

            if sent.strip() != '':
                tok = Token(sent+' ::', Location(i, unit='s', source=source))
                tokens.append(tok)
                i += 1
        return tokens
```

Tagger.py

```
class myNN_CD_Tagger(SequentialTagger):
    """
    A "default" tagger, which will assign the tag C{"CD"} to numbers,
    and C{"NN"} to anything else.  This tagger expects token types to
    be C{strings}s.
    """
    def __init__(self): pass

    def next_tag(self, tagged_tokens, next_token):
        # Inherit docs from SequentialTagger
        assert _chktype(1, tagged_tokens, [Token], (Token,))
        assert _chktype(2, next_token, Token)

        if re.match(r'^[0-9]+(.[0-9]+)?$', next_token.type()):
            return 'CD'
        elif re.match(r'[0-9]+(,[0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'\S+[0-9]+[M]', next_token.type()):
            return 'CD'
        elif re.match(r'[0-9]+(;[0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'[0-9]+(,[0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'[0-9]+(;[0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'\S+[0-9]+(;[0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'\S+[0-9]+(;[0-9]+)+(///+)([0-9]+)?', next_token.type()):
            return 'CD'
        elif re.match(r'\S+[0-9]+(,[0-9]+)+(///+)([0-9]+)?', next_token.type()):
            return 'CD'
        else:
            return 'NN'
```

Appendix H -

Velocity template code for printing RDF output

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<!-- Output from textfile: $fileName -->
<!-- RDF file: $myRDF_file -->
<!DOCTYPE rdf:RDF
      [<!ENTITY kb 'http://protege.hia.no/ebjoraa/kb#'>
       <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
       <!ENTITY s 'http://protege.hia.no/ebjoraa/article/sentence#'>
       <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'>     ]>
       <rdf:RDF xmlns:kb="&kb;"
              xmlns:rdf="&rdf;"
              xmlns:s="&s;"
              xmlns:rdfs="&rdfs;">
<rdf:Description about="$URL"
           #if ($pubDate)
            s:published="$pubDate"
           #end
           #if ($Language)
            s:language="$Language"
           #end
           #if ($Source)
            s:source="$Source"
           #end
           #if ($Subject)
            s:subject="$Subject"
           #end
           >
        #set ($sid = 1)
        #foreach($s in $sentences)
         <kb:part-of>
         <rdf:Description about="http://protege.hia.no/ebjoraa/article#sentenceID$sid"
           #if ($s.SText)
            s:text="$s.SText"
           #end
         >
           <s:has-CA>
          #set ($caid = 1)
```

```
#foreach($c in $s.CuCa)
 <rdf:Description about=
  #if ($caid == 1)
    #if ($c.CU == "0000")
     "No financial figures found in sentence!">
    #else
        "http://protege.hia.no/ebjoraa/sentenceID$sid/currencyAmountID/$caid">
      <s:currencyUnit>$c.CU</s:currencyUnit>
      <s:currencyAmount>$c.CA</s:currencyAmount>
    #end
  #else
      "http://protege.hia.no/ebjoraa/sentenceID$sid/currencyAmountID/$caid">
      <s:currencyUnit>$c.CU</s:currencyUnit>
      <s:currencyAmount>$c.CA</s:currencyAmount>
  #end
 </rdf:Description>
  #set ($caid = $caid + 1)
#end
 </s:has-CA>
 <s:has-FD>
#set ($caid = 1)
 <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID$sid/finDescr">
  #if ($s.predict)
    <kb:predict>$s.predict</kb:predict>
  #end
    #if ($s.ProfitLoss)
    <kb:ProfitLoss>$s.ProfitLoss</kb:ProfitLoss>
    #end
  #if ($s.financeAction)
    <kb:financeAction>$s.financeAction</kb:financeAction>
  #end
  #if ($s.annual_report)
    <kb:annual_report>$s.annual_report</kb:annual_report>
  #end
  #if ($s.financePred)
    <kb:predict>$s.financePred</kb:predict>
  #end
  #if ($s.financeOutput)
    <kb:financeOutput>$s.financeOutput</kb:financeOutput>
```

```
            #end
            #if ($s.measurePositive)
              <kb:measurePositive>$s.measurePositive</kb:measurePositive>
            #end
            #if ($s.measureNegative)
              <kb:measureNegative>$s.measureNegative</kb:measureNegative>
            #end
            #if ($s.accountAnnualFact)
              <kb:accountAnnualFact>$s.accountAnnualFact</kb:accountAnnualFact>
            #end
            #if ($s.company)
              <kb:company>$s.company</kb:company>
            #end
            #if ($s.financeInput)
              <kb:financeInput>$s.financeInput</kb:financeInput>
            #end
            #if ($s.financialPapers)
              <kb:financialPapers>$s.financialPapers</kb:financialPapers>
            #end
            #if ($s.exchangeWords)
              <kb:exchangeWords>$s.exchangeWords</kb:exchangeWords>
            #end
            #if ($s.payMoney)
              <kb:payMoney>$s.payMoney</kb:payMoney>
            #end
            #if ($s.period)
              <kb:periode>$s.period</kb:periode>
            #end
            #if ($s.periode_info)
              <kb:periode_info>$s.periode_info</kb:periode_info>
            #end
           </rdf:Description>
           </s:has-FD>
         </rdf:Description>
          </kb:part-of>
         #set ($sid = $sid + 1)
      #end
 </rdf:Description>
</rdf:RDF>
```

Appendix I -

Velocity template code for printing XBRL output

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!--  Date/time article created: $pubDate -->

<group xmlns="http://www.xbrl.org/2001/instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:iascf-pfs="http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15"
  xmlns:iso4217="http://www.xbrlSolutions.com/taxonomies/iso4217/2002-06-30"
  xsi:schemaLocation="http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15
  ias-ci-pfs-2002-11-15.xsd
  http://www.xbrlSolutions.com/taxonomies/iso4217/2002-06-30
  http://www.xbrlsolutions.com/taxonomies/iso4217/2002-06-30/iso4217.xsd">


 #set ($id = 1)
 #foreach($s in $sentences)

            #if($s.ProfitLoss)
            <!-- Profit before/after tax -->
             #foreach($c in $s.CuCa)
            #if ($s.tax == "before_tax")
                <iascf-pfs:ProfitLossBeforeTax numericContext="Current_ForSentence$id">$c.CA</iascf-
pfs:ProfitLossBeforeTax>
            #elseif($s.tax == "after_tax")
              <iascf-pfs:ProfitLossAfterTax numericContext="Current_ForSentence$id">$c.CA</iascf-pfs:ProfitLossAfterTax>

            #else
              <iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence$id" comment="Before or after tax is not
specified!">$c.CA</iascf-pfs:ProfitLossOperations>
                #end
       #set ($caid = $caid + 1)
      #end
              #end

 <numericContext id="Current_ForSentence$id" cwa="false">
 <entity>
  <identifier scheme="$URL">$Source</identifier>
  </entity>
 <period>
 #if ($s.period)
  #if ($s.period == "fourth_quarter")
```

```
   <duration>P1Q</duration>
   <endDate>$Pyear -12-31</endDate>
  #end
 #else
 <instant>Period not given!</instant>
 #end
  </period>
 <unit>
             #set ($caid = 1)
             #foreach($c in $s.CuCa)
                      <measure>iso4217:$c.CU</measure>
             #set ($caid = $caid + 1)
             #end

  </unit>
  </numericContext>


#set ($id = $id + 1)
#end
</group>
```

Appendix J - Example of XML input file of article from Intermedium's agent

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<Finance>
      <article>
            <Source>Reuters</Source>
            <URL>http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html</URL>
            <language>English</language>
            <Subject>Advanced Marketing posts quarterly loss</Subject>
            <pubDate>20030522</pubDate>
            <Text>

NEW YORK, May 22 (Reuters) - Advanced Marketing Services Inc. (NYSE:MKT - News), a distributor of books to warehouse
clubs, on Thursday reported a quarterly loss after a high number of books were returned by customers late in the
quarter.

For the fiscal fourth quarter ended March 31, Advanced Marketing reported a loss of $4.4 million, or 23 cents per share,
reversing a profit of $3 million, or 15 cents per share, a year earlier.

The result was in line with estimates for a loss of 20 to 24 cents per share the company provided earlier this month
when it said results for the quarter would miss its targets. That warning triggered a sharp sell-off in the San Diego-
based company's stock.

Looking ahead, the company held to its previous estimate of fiscal 2004 earnings of between 90 cents and $1.05 per
share, up from the prior year's 57 cents per share.

      </Text>
  </article>
</Finance>
```

Appendix K - RDF output based on article in Appendix J

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
- <!-- Output from textfile: inputfile.xml   -->
- <!--  RDF file: Finance.rdf   -->
<!DOCTYPE rdf:RDF (View Source for full doctype...)>
- <rdf:RDF xmlns:kb="http://protege.hia.no/ebjoraa/kb#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://protege.hia.no/ebjoraa/article/sentence#"
xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
- <rdf:Description
about="http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html"
s:published="20030522"
s:language="English"
s:source="Reuters"
s:subject="Advanced Marketing posts quarterly loss">
- <kb:part-of>
- <rdf:Description about="http://protege.hia.no/ebjoraa/article#sentenceID1" s:text="NEW YORK, May 22 (Reuters) - Advanced Marketing Services Inc; (NYSE:MKT - News), a distributor of books to warehouse clubs, on Thursday reported a quarterly loss after a high number of books were returned by customers late in the quarter; For the fiscal fourth_quarter ended March 31, Advanced Marketing reported a loss of $4,4M, or 23cent per share, reversing a profit of $3M, or 15cent per share, a year earlier ::">
- <s:has-CA>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID1/currencyAmountID/1">
<s:currencyUnit>usd</s:currencyUnit>
<s:currencyAmount>4,4M,</s:currencyAmount>
</rdf:Description>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID1/currencyAmountID/2">
<s:currencyUnit>cent</s:currencyUnit>
<s:currencyAmount>23</s:currencyAmount>
</rdf:Description>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID1/currencyAmountID/3">
<s:currencyUnit>usd</s:currencyUnit>
<s:currencyAmount>3M,</s:currencyAmount>
</rdf:Description>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID1/currencyAmountID/4">
```

```
  <s:currencyUnit>cent</s:currencyUnit>
  <s:currencyAmount>15</s:currencyAmount>
    </rdf:Description>
    </s:has-CA>
- <s:has-FD>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID1/finDescr">
  <kb:ProfitLoss>profit</kb:ProfitLoss>
  <kb:accountAnnualFact>reported</kb:accountAnnualFact>
  <kb:financialPapers>share,</kb:financialPapers>
  <kb:periode>fourth_quarter</kb:periode>
    </rdf:Description>
    </s:has-FD>
    </rdf:Description>
    </kb:part-of>
- <kb:part-of>
- <rdf:Description about="http://protege.hia.no/ebjoraa/article#sentenceID2" s:text="The result was in line with estimates
    for a loss of 20 to 24cent per share the company provided earlier this month when it said results for the quarter would
    miss its targets ::">
- <s:has-CA>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID2/currencyAmountID/1">
  <s:currencyUnit>cent</s:currencyUnit>
  <s:currencyAmount>24</s:currencyAmount>
    </rdf:Description>
    </s:has-CA>
- <s:has-FD>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID2/finDescr">
  <kb:predict>estimates</kb:predict>
  <kb:ProfitLoss>loss</kb:ProfitLoss>
  <kb:accountAnnualFact>results</kb:accountAnnualFact>
  <kb:financialPapers>share</kb:financialPapers>
    </rdf:Description>
    </s:has-FD>
    </rdf:Description>
    </kb:part-of>
- <kb:part-of>
- <rdf:Description about="http://protege.hia.no/ebjoraa/article#sentenceID3" s:text="That warning triggered a sharp sell-off
    in the San Diego-based company's stock ::">
```

```
  <s:has-CA />
- <s:has-FD>
  <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID3/finDescr" />
    </s:has-FD>
    </rdf:Description>
    </kb:part-of>
- <kb:part-of>
- <rdf:Description about="http://protege.hia.no/ebjoraa/article#sentenceID4" s:text="Looking ahead, the company held to its
    previous estimate of fiscal 2004 earnings of between 90cent and $1,05 per share, up from the prior year's 57cent per
    share. ::">
- <s:has-CA>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID4/currencyAmountID/1">
  <s:currencyUnit>cent</s:currencyUnit>
  <s:currencyAmount>90</s:currencyAmount>
    </rdf:Description>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID4/currencyAmountID/2">
  <s:currencyUnit>usd</s:currencyUnit>
  <s:currencyAmount>1,05</s:currencyAmount>
    </rdf:Description>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID4/currencyAmountID/3">
  <s:currencyUnit>cent</s:currencyUnit>
  <s:currencyAmount>57</s:currencyAmount>
    </rdf:Description>
    </s:has-CA>
- <s:has-FD>
- <rdf:Description about="http://protege.hia.no/ebjoraa/sentenceID4/finDescr">
  <kb:predict>estimate</kb:predict>
  <kb:financialPapers>share</kb:financialPapers>
    </rdf:Description>
    </s:has-FD>
    </rdf:Description>
    </kb:part-of>
    </rdf:Description>
    </rdf:RDF>
```

Appendix L - XBRL output based on article in Appendix J

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
- <!--  Date/time article created: 20030522  -->
- <group xmlns="http://www.xbrl.org/2001/instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:iascf-pfs="http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15"
  xmlns:iso4217="http://www.xbrlSolutions.com/taxonomies/iso4217/2002-06-30"
  xsi:schemaLocation="http://www.xbrl.org/taxonomy/int/fr/ias/ci/pfs/2002-11-15 ias-ci-pfs-2002-11-15.xsd
  http://www.xbrlSolutions.com/taxonomies/iso4217/2002-06-30 http://www.xbrlsolutions.com/taxonomies/iso4217/2002-06-
  30/iso4217.xsd">
- <!--  Profit before/after tax  -->
<iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence1" comment="Before or after tax is not
  specified!">4,4M,</iascf-pfs:ProfitLossOperations>
<iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence1" comment="Before or after tax is not
  specified!">23</iascf-pfs:ProfitLossOperations>
<iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence1" comment="Before or after tax is not
  specified!">3M,</iascf-pfs:ProfitLossOperations>
<iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence1" comment="Before or after tax is not
  specified!">15</iascf-pfs:ProfitLossOperations>
- <numericContext id="Current_ForSentence1" cwa="false">
- <entity>
<identifier scheme="http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html">Reuters</identifier>
  </entity>
- <period>
<duration>P1Q</duration>
<endDate>2002 -12-31</endDate>
  </period>
- <unit>
<measure>iso4217:usd</measure>
<measure>iso4217:cent</measure>
<measure>iso4217:usd</measure>
<measure>iso4217:cent</measure>
  </unit>
  </numericContext>
- <!--
 Profit before/after tax
```

```
    -->
  <iascf-pfs:ProfitLossOperations numericContext="Current_ForSentence2" comment="Before or after tax is not
    specified!">24</iascf-pfs:ProfitLossOperations>
- <numericContext id="Current_ForSentence2" cwa="false">
- <entity>
  <identifier scheme="http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html">Reuters</identifier>
    </entity>
- <period>
  <instant>Period not given!</instant>
    </period>
- <unit>
  <measure>iso4217:cent</measure>
    </unit>
    </numericContext>
- <numericContext id="Current_ForSentence3" cwa="false">
- <entity>
  <identifier scheme="http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html">Reuters</identifier>
    </entity>
- <period>
  <instant>Period not given!</instant>
    </period>
  <unit />
    </numericContext>
- <numericContext id="Current_ForSentence4" cwa="false">
- <entity>
  <identifier scheme="http://biz.yahoo.com/rc/030522/retail_advancedmarketing_earns_1.html">Reuters</identifier>
    </entity>
- <period>
  <instant>Period not given!</instant>
    </period>
- <unit>
  <measure>iso4217:cent</measure>
  <measure>iso4217:usd</measure>
  <measure>iso4217:cent</measure>
  </unit>
  </numericContext>
  </group>
```